Pythonでアプリを作成してみる

～　Kivyライブラリーによるデスクトップアプリから
Androidアプリの作成について～

オカザキ（Studio Overdrive）

内容

この本の内容について 4
本書の構成について 4
この本の対象レベル 6
検証環境 6
参考資料について 7
サンプルコードについて 7
Kivy とは 7
Kivy のインストールについて 7
Windows 8
Mac 8
Ubuntu 8

１章 ～Kv Language の基本～ 10
あらまし 10
今回作成するプログラムについて 10
参考資料 10
プログラムについて 11
検証環境 11
プログラムのファイル名について 11
実際の説明 12
1. 空の画面を表示する 12
2. App およびラベルの表示 13
3. App およびラベルの表示（Kv Language を用いたもの） 15
4. 複数のラベルの表示 18
5. 複数のラベルの表示（縦並びにする） 20
6. サイズの調整 22
7. サイズの調整 2 24
8. 文字の色とラベルの色の変更 26
9. ボタンの追加 30
10. ボタンを押した場合の機能の追加 32
11. 複数のボタンを追加した場合 36
12. レイアウトを変更 41
13. クリック時の機能追加(文字の色を変更する) 45
14. 日本語の表示（フォントの追加） 50
15. 起動時の解像度を変更およびパラメータの使いまわしについて 57
16. 画像の表示 64

17. 画像の表示（ボタンによる画像切り替え） 68
まとめ 72
参考　ファイルを選択する 73
参考　日本語の入力について（注意） 74
参考　kv ファイルの値を Python 側で取得する方法 77
参考　Kivy 側で Python ファイルの関数の呼び方 78
参考　GridLayout の使いかた 81
参考　設定画面 84
参考　起動オプション 84
Kivy を使用して作成に向いているアプリ 85

２章　～電卓を作成する～ 86
あらまし 86
作成するもの 86
あらたに習得する内容 86
参考リンク 87
検証環境 87
実行結果 88
コード 89
Kv ファイル 92
解説 99
起動時のレイアウトについて 99
TextInput について 100
GridLayout について 101
ActionBar について 102
atlas について 103
Kv ファイルでの複数行実行について 105
get_color_from_hex()による色の取得について 106
解説 2 106
widget の切り替え 106
まとめ 108

３章　～WebAPI との連携（リクエストの送受信から結果表示まで）～ 109
あらまし 109
作成するもの 109
起動画面（一覧表示） 109
詳細画面 110
実際の検索結果 110

詳細画面（検索結果選択時） 111
一覧画面（検索語、詳細画面から遷移時） 112
あらたに習得する内容 112
参考リンク 113
検索する内容について 113
検証環境 113
コード 114
Kv ファイル 117
解説 121
「BookSearchRoot」widget について 123
Carousel について 124
アクションバー 126
SearchBookForm について 127
「BookInfo」widget について 134
まとめ 138
参考　ファイルのパッケージ化について 138

４章　～Android での実行～ 139
あらまし 139
作成するもの 139
使用するコード 139
Kivy を Andorid 端末で実行する方法 139
実行する Python 環境について 140
実際に検証に使用した Android 端末について 140
Android で Kivy を実際に動かす 140
Kivy Launcher について 141
Kivy Lancher の使い方 141
前回のアプリを動してみる 145
Kivy の UrlRequest の使い方 145
実行結果 147
参考　Android 特有の API を実行する方法 147
参考　iOS へ対応について 148
参考 Python の外部ライブラリを使用するには 149
参考 Kivy で作成されたアプリについて 149
まとめ 149
全体のまとめ 151

この本の内容について

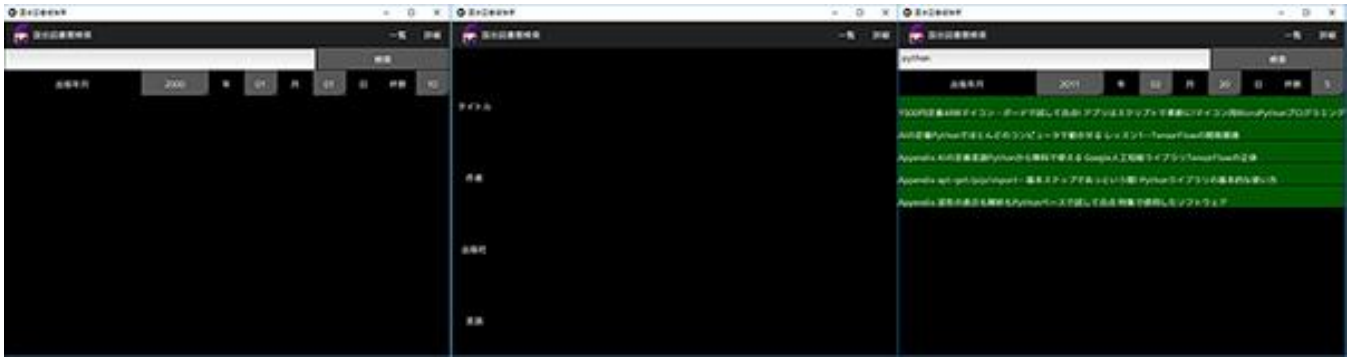
この本は、プログラミング言語 Python を使用しアプリを作成してみる本です。Python はここ 2,3 年、機械学習の流行の関係で本が多数出版されました。しかしながら多くは機械学習関係の本か文法的な入門書的な本が多く、応用的な書籍が少ないです。
この本では、Kivy というライブラリを使用して、Kivy の基本的な使い方の紹介から、実際に Android で実行するプログラムを作成します。
最後まで読み進めれば Kivy を使用して Python で GUI アプリの作成が可能になります。

本書の構成について

1 章の「Kv Language の基本」では主にラベルでの表示から、日本語の表示、最終的には複数のボタンを配置して、ボタンによって表示される内容を変えることを学習します。

2 章の「電卓を作成する」では 1 章の内容を踏まえて実際に画面 1 枚のアプリの作成から、ボタンによるレイアウトの変更を学習します。

3 章の「WebAPI との連携（リクエストの送受信から結果表示まで）」では国立国会図書館サーチの外部提供インタフェース API（API）（http://iss.ndl.go.jp/information/api/）を用いて検索、結果を表示するプログラミングの書き方、マルチ画面の使用方法を学習します。



4 章は、3 章で作成したプログラムを元に andoriod を用いて実機で動作する事を学習します。

この本の対象レベル

Python の入門書を一冊読んで文法的に基本的な内容を理解している人を対象にしています。
説明は主に Kivy の使い方の説明に言及しており、Python の文法的な説明はしておりません。

検証環境

この内容の検証環境は以下の通りです。

[ハード]

- OS:Windows10 64bit
- エディター：Visual Studio 2015(PTVS)、サクラエディタ

※Windows での検証ですが、コード自体は Mac、Ubuntu でも実行できます。

[ソフト]

- Kivy:1.9.1
- Python3.4 or Python2.7（どちらも 32bit）

android の検証環境：

- Nexus (OS Android6.01)
- ZenFone 3 Laser(OS Android6.01)

参考資料について

- 公式サイト(https://kivy.org/#home)
- 有志による非公式翻訳サイト(https://pyky.github.io/kivy-doc-ja/)

また web 上の記事を適宜紹介します。

サンプルコードについて

検証用のコードは以下からダウンロード可能です。

- github(https://github.com/okajun35/Kivy_practice)

Kivy とは

Python の NUI(Natural User Interface)でのマルチタッチアプリケーション開発のためのオープンソースライブラリです。
Python と Cython で作成されており、OpenGL ES 2 をベースにしています。
Windows, OS X, Linux, Android iOS, そして、RasberryPI 上でほぼ同一のコードでの実行が可能です。
ライセンスは MIT ライセンスなので商用利用も可能です

Kivy の利点とライセンスについて詳しく知りたい方は公式サイトで紹介されていますのでそちらをご覧ください。

- User's Guide(翻訳済み) » Philosophy(翻訳済み)(https://pyky.github.io/kivy-doc-ja/philosophy.html)
- Programming Guide(翻訳済み) » Package licensing (翻訳済み)(https://pyky.github.io/kivy-doc-ja/guide/licensing.html)

Kivy のインストールについて

基本は公式サイトのインストールを読んでやれば大丈夫です。
pip を使ってのインストールになります。
Python のバージョンは Python3.4 または Python2.7 になります。

- User's Guide(翻訳済み) » Installation(翻訳済み)(https://pyky.github.io/kivy-doc-ja/installation/installation.html)

Windows

Windows のインストールは以下になります。

```
python -m pip install --upgrade pip wheel setuptools
python -m pip install docutils pygments pypiwin32 kivy.deps.sdl2 kivy.deps.glew
python -m pip install kivy.deps.gstreamer --extra-index-url
https://kivy.org/downloads/packages/simple/
python -m pip install kivy
```

なお Windows で 64bit の場合は公式サイト通りだとインストールできません。
その際は非公式バイナリ置き場(http://www.lfd.uci.edu/~gohlke/pythonlibs/#kivy) から whl をダウンロードしてインストールしてください。
また Python2 系の場合は gstreamer のインストールに失敗することがあります。
gstreamer は音、映像を扱うライブラリで、Kivy では必須のライブラリではありません。本書では音、映像を特に使用しないのでインストールできなくても問題はないです。

- Python3.4 ＋ Visual Studio で Kivy の開発環境を作成する（http://qiita.com/dario_okazaki/items/fe2682fbd88e971c2ef2）

Mac

Mac のインストールは以下になります。

```
$ brew install sdl2 sdl2_image sdl2_ttf sdl2_mixer gstreamer
$ pip install -I Cython==0.23
$ USE_OSX_FRAMEWORKS=0 pip install kivy
```

Ubuntu

Ubuntu のインストールは以下になります。

```
sudo add-apt-repository ppa:kivy-team/kivy
sudo apt-get update
sudo apt-get install python-kivy
```

1章 ～Kv Language の基本～

あらまし

Python の GUI ライブラリを作る Kivy(https://kivy.org/#home) ですが、日本では知名度はさほどではありません。理由として一番多いのはインストールしたのは良いが日本語でまとまった情報があまりなく、どう使用していいかわからないという声をよく聞きます。また Kivy にはレイアウトを記述する際に Python によるコード入力とは別に Kv Language と呼ばれる、CSS に似た Kivy 独時の言語を用いてレイアウトを作成できますが、情報が少なく慣れるまで時間がかかります。
ここでは Kv Language を用いた簡単なプログラムを紹介して Kv Language の基本的な使い方を紹介します。

今回作成するプログラムについて

今回作成するプログラムは以下の通りです。

画像左端が起動直後の状態です、下にある「朝」、「昼」、「夜」のボタンをクリックすることでそれぞれ、「おはよう」、「こんにちは」、「こんばんは」の文字が表示されます。
内容としては、Kv Language によるレイアウトの仕方とボタンをクリックした際の機能の実行の仕方の紹介です。

参考資料

資料ですが Kivy の公式マニュアル（https://kivy.org/docs/ ） の Programming Guide から「Kivy Basics」から「Kv Language」 の内容になります。ただし英語で書かれており理解が難しいので有志が翻訳した非公式※の日本語に翻訳（https://pyky.github.io/kivy-doc-

ja/ ）したものを引用します。
※本家のサイトに事前に連絡して翻訳作業自体は問題がないことを確認しております。

プログラムについて

Kivy を使用したコードの書き方ですが、色々な書き方があります。あくまでもこれは一例です。

検証環境

検証環境は以下の通りです。

- OS: Windows10 64bit
- Kivy:1.9.1
- Python3.4※

※Python2 系でもコードは実行できますが、日本語表示の箇所でエラーになります。その際は、コードを変更してみてください。（例:'日本語'→u'日本語'）

プログラムのファイル名について

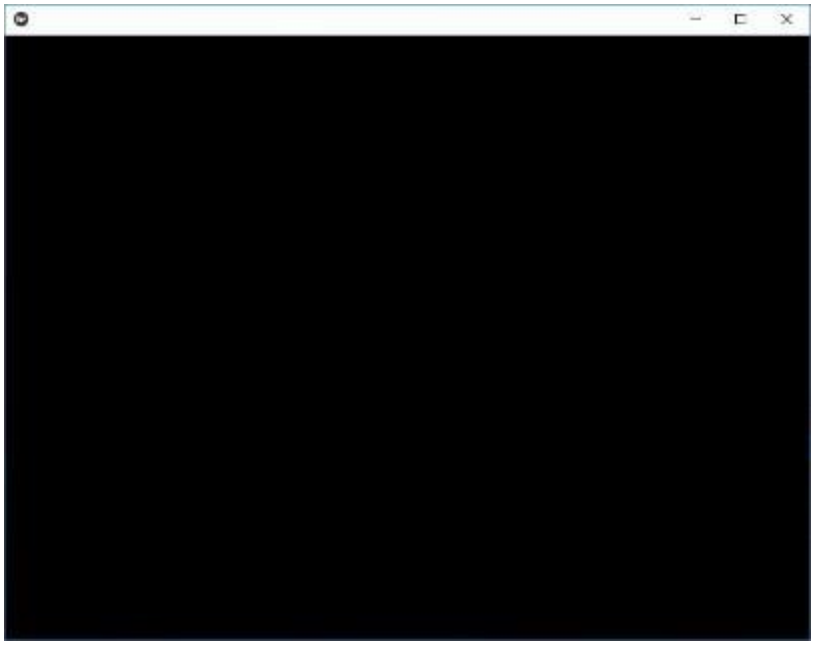
Python ファイル名は、main.py、Kv Language ファイルは test.kv として保存してください。

実際の説明

以下、実際にコードと結果を載せていきます。

1. 空の画面を表示する

実行結果



コード

```
#-*- coding: utf-8 -*-

from kivy.app import App

App().run()
```

Kv ファイル

今回は使用しません

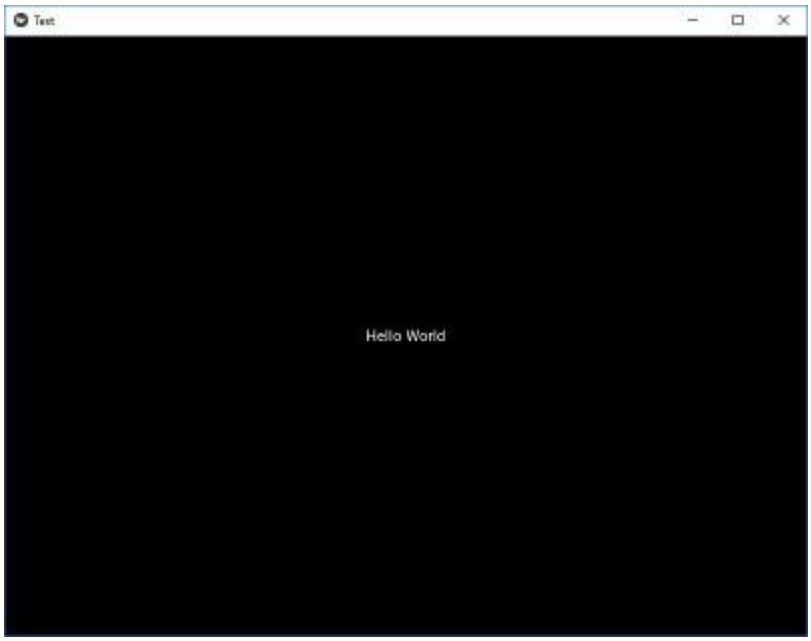
解説

Kivy の基本的な仕組みは Programming Guide(翻訳済み) » Kivy Basics(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/basic.html）を参考にしてください。

ここでは、App クラスは、Kivy アプリを作成するための基本になるクラスです。
基本的な使い方は自分のアプリを作るときに App クラスを継承してサブクラスを作り、そのラベルやボタンなどを追加することを覚えていれば十分です。

2. App およびラベルの表示

実行結果



コード

```
#-*- coding: utf-8 -*-

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

class TestApp(App):
```

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

TestApp().run()
```

Kv ファイル

今回は使用しません

解説

App クラスを継承して TestApp クラスを作成しました。 さらに build 関数を実行して、作成時に Label()を返すようにしました。 実行すると、画面中央に「Hello world」の文字が表示されたラベルが画面全体に作成されてます。 Label()は Kivy の widget（ グラフィカルユーザインタフェースを構成する部品要素、およびその集まり）の一つです。Kivy はパーツ、レイアウトなどの widget を組み合わせて GUI を作成します。

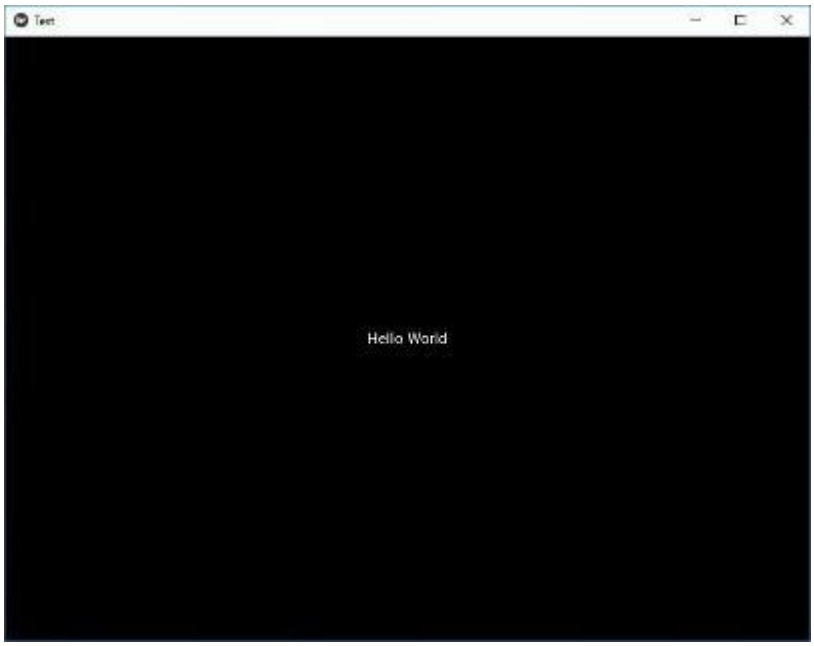
またウィンドウの右上のタイトルに「Test」と表示されています。kivy ではタイトルは App クラスを継承したサブクラスの名のうち App の前までの名前が表示されます。（例：TestApp→Test が表示されます）

さらに詳しく知りたい方は API リファレンス（英語）を参考にしてください。

- App(API Reference)（https://kivy.org/docs/api-kivy.app.html）
- Label(API Reference)（https://kivy.org/docs/api-kivy.uix.label.html）

3. App およびラベルの表示（Kv Language を用いたもの）

実行結果



コード

```
#-*- coding: utf-8 -*-
from kivy.app import App

class TestApp(App):
    pass

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

Kv ファイル

```
Label:  # add comment
    text: "Hello World"
```

解説

2 と違うのは、main.py 側ではサブクラス TestApp を作成しただけで、内部では何もしていません。代わりに新規に test.kv という Kv(Kivy language)ファイルを新規に作っています。Kv ファイルですが kivy ではタイトルは App クラスを継承したサブクラスの名のうち App の前までと同じ**先頭が小文字のファイル名**が対応します。ここではサブクラス名が TestApp なので、test.kv が対応します。

Kivy language ですが次の特徴を持っています。

- html の CSS によく似ています。
- 書き方としては、追加する値、クラスに関して、＜変数名 or 関数名＞: ＜値＞　と書きます。「:」ですが前の＜変数名＞との間には**スペースを開けないでください**。
- Python と同じくインデントでスペースまたはタブでできます。注意するのはインデントは**ファイルの頭から最初に見つけたスペースの数またはタブの数できまります**。例えば、kv ファイルの頭で半角スペース一文字を入れて改行すると、以降インデントは半角スペース一文字に設定されますので、半角スペース 4 文字でインデントを付けるとエラーになります。
- 「#」でコメントを付けられます。python のように'"＜コメント＞"'は不可です

今回は、test.kv 内で

```
Label:  # add comment
    text: "Hello World"
```

と宣言することで、Python ファイル内で Label()関数の text 引数に"Hello world"を指定して実行（Label(text='Hello World')）したことと同じになります。

Kv Language については、ここでも解説しますが Programming Guide(翻訳済み) » Kv language(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/lang.html）を一読することをお勧めします。

参考

- Kivy Language(API Reference)（https://kivy.org/docs/api-kivy.lang.html）

[Note] Kv Language の使用について

Kv Language を使用しなくても、他の GUI ライブラリのように Python でコードを書いてレイアウトを作成する事は可能です。日本でも Python コードのみでも書かれている方もいます。

- 参考　Python の GUI ライブラリ Kivy によるクロスプラットフォーム GUI アプリ作成入門（http://myenigma.hatenablog.com/entry/2016/05/10/221433）

ただし Kv Language を使用することで**レイアウトと機能の記述を分けることができます**。これにより機能追加などをしたい場合やレイアウトの変更をしたい場合にコードの理解がしやすくなります。例えばアプリを作成して、ユーザーに配布した場合、機能の変更がないがレイアウトの変更をしたいという場合には、kv ファイルをユーザーが直接変更してレイアウトを自分好みに変更することも可能です。

[Note] Kv Language を書くファイルについて

Kv Language ですが、どのファイルに書くかですが 3 つのパターンが選べます。

1. Python ファイルに直書きする
2. App のサブクラス名に対応した kv ファイル（拡張子.kv）を作成して記述する
3. Python ファイル内で対応する Kv ファイルを指定して、その Kv ファイルを記載する

1 の Python ファイルに直書きする方法ですが、Builder クラスを import して記述します。
書き方は以下の通りです。

```
from kivy.lang import Builder
Builder.load_string("""
<App 名>:
内容
""")
```

2 の方法はすでに取り上げているので省略します。
3 の Python ファイル内で対応する Kv ファイルを指定する方法ですが、書き方は以下の通りです。

```
from kivy.lang import Builder
Builder.load_file('file.kv')
```

この場合 file.kv がその Python ファイルに対応する Kv ファイルになります。
通常は 1 か 2 の方法を用いて記述することが多いです。
1 と 2 のどちらがいいかはケースバイケースかと思います。

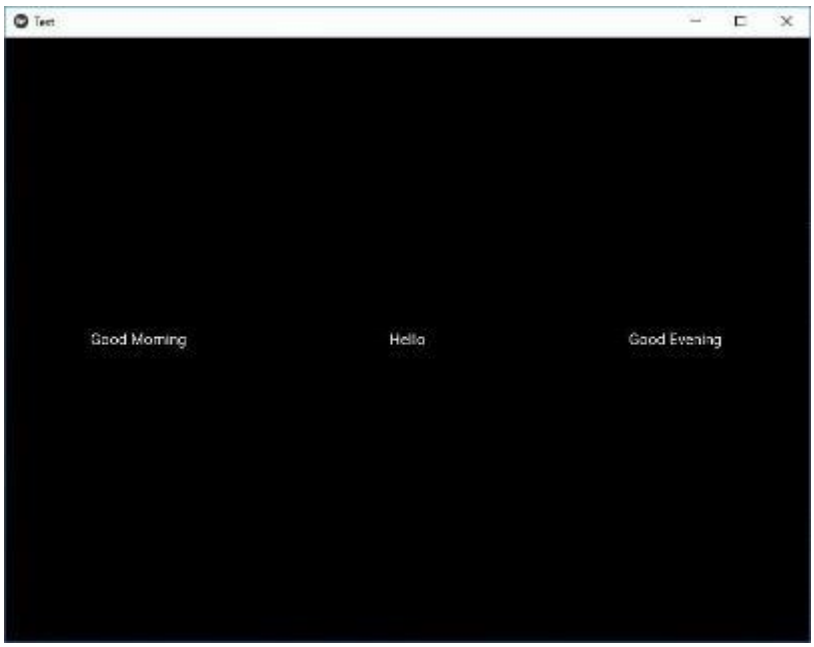
参考

- Kv ファイルのロード（http://code.tiblab.net/python/kivy/kv_load）

また、今回は取り上げませんが Kv ファイルを widght の名前と同じ名前の kv ファイルを作成することで複数ファイルに分割することも可能のようです。（未検証）

4. 複数のラベルの表示

実行結果



コード

（3.の内容と変更はありません。）

Kv ファイル

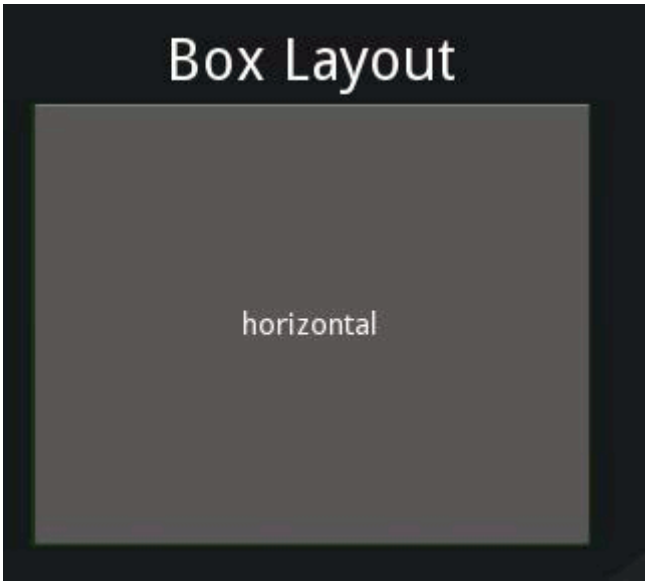
Kv Language は以下の通りです。

```
BoxLayout:
    Label:
        text: "Good Morning"
    Label:
        text: "Hello"
    Label:
        text: "Good Evening"
```

解説

今回は、Label を 3 つ作成して横に等間隔に並べています。レイアウトの設定に**「BoxLayout」**を使用しています。kivy には widget を並べるためのレイアウトがいくつか用意されています。「BoxLayout」は**複数の widget を縦または横にならべるためのレイアウト**です。

公式マニュアルから転載



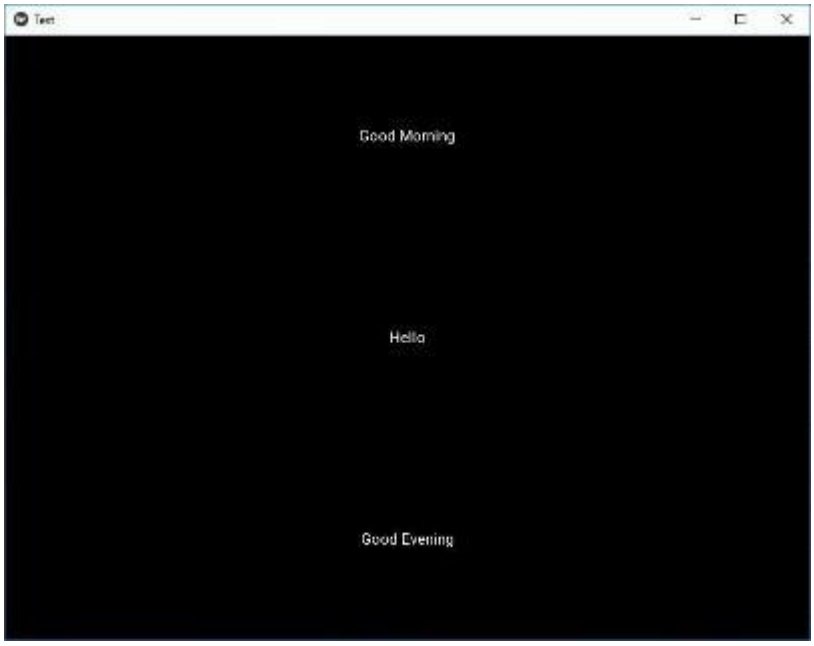
Kivy はこのレイアウトを一つまたは複数を使用することでレイアウトを作成していきます。レイアウトの種類ですが BoxLayout の他に「FloatLayout」、「RelativeLayout」、「GridLayout」、「PageLayout」、「ScatterLayout」、「StackLayout」がありますがまずは「BoxLayout」でレイアウトを作成することをお勧めします。

参考

- Getting Started(翻訳済み) » Layouts(翻訳済み)（https://pyky.github.io/kivy-doc-ja/gettingstarted/layouts.html）
- Programming Guide(翻訳済み) » Widgets(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/widgets.html）
- Box Layout(API Reference)（https://kivy.org/docs/api-kivy.uix.boxlayout.html）

5. 複数のラベルの表示（縦並びにする）

実行結果



コード

（3.の内容と変更はありません。）

Kv ファイル

Kv Language は以下の通りです。

```
BoxLayout:
    orientation: 'vertical'     # 'horizontal'だと横一列

    Label:
        text: "Good Morning"
    Label:
        text: "Hello"
    Label:
        text: "Good Evening"
```

解説

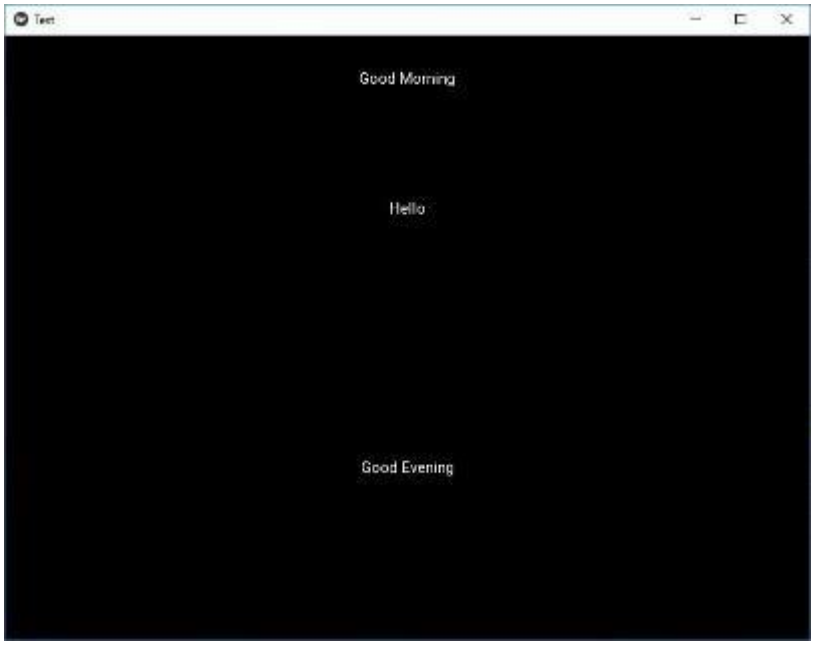
4.の場合ではラベルは横に等間隔に並んでいましたが、今回は縦に 3 つ並んでいます。前との違いは Kv ファイルに `orientation: 'vertical'` の記述がある点です。`orientation` は BoxLayout のオプションの一つで widget の並びを設定します。値は 2 種類あり、**「horizontal」だと横一列、「vertical」だと縦一列**に widget の並びが設定されます。4.の場合に orientation の記載がないのに横一列に並んでいるのは orientation のデフォルトの設定が「horizontal」になっているためです。

[Note] Kivy の座標系について

今回の並びをみているとわかるのですが、Kivy の原点座標は**左下**になります。これが Kivy のとっつきにくい点の一つになっています。慣れると問題がないのですが、多くの GUI のソフトと違い Kivy は原点が左下にあると覚えておいてください。

6. サイズの調整

実行結果



コード

（3.の内容と変更はありません。）

Kv ファイル

Kv Language は以下の通りです。

```
BoxLayout:
    orientation: 'vertical'     # 'horizontal'だと横一列

    Label:
        text: "Good Morning"
        size_hint_y: 0.5
    Label:
        text: "Hello"
        size_hint_y: 1
    Label:
```

```
        text: "Good Evening"
        size_hint_y: 2
```
実行結果は以下の通りです。

解説

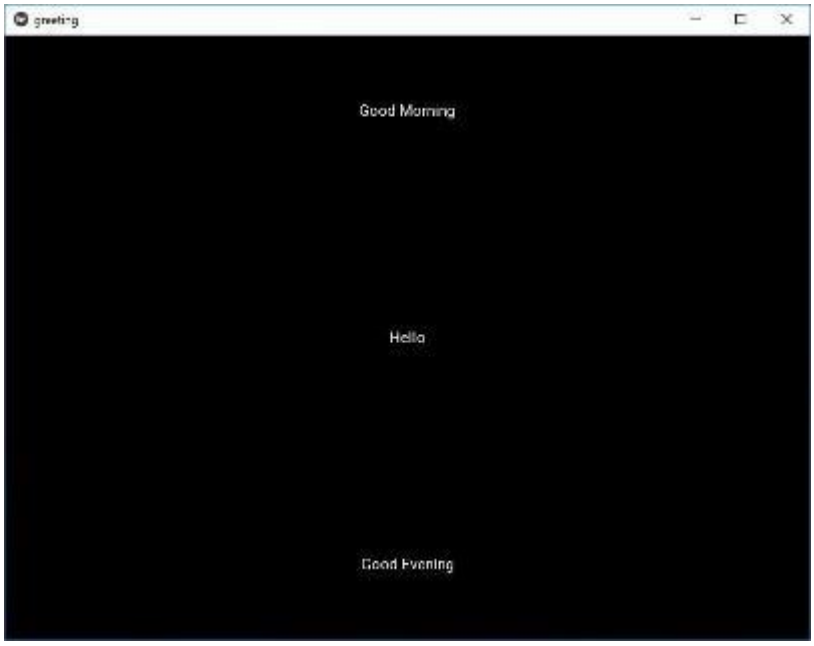
`size_hint_y: <値>`を新たにパラ―メータに加えています。それにより Label の位置が変わっています。**size_hint_x** に値を加えることで x 方向の大きさを、**size_hint_y** に値を加えることでy方向の大きさが変わります。デフォルトでは両方とも 1 に設定されています。 大きさは画面全体の大きさ分の各比率になります。 例として今回の場合は"Good Morning" のラベルのサイズですが、 Y 方向の比率 = (Good Morning の値)/(全部の size_hint の値)=0.5/(0.5+1+2)=14% になり、画面全体で 14%のサイズになります。
size_hint 以外に、座標を画面の比率から設定する pos_hint という設定もあります。ちなみに Kivy では座標を比率ではなく座標で指定することもできますがあまり推奨されていません。理由は画面サイズを自由に変更できるために widget を座標で指定すると位置が固定のためにきれいに表示されないからです。

Kivy で位置を指定に関しては色々な指定方法がありますが、基本は **size_hint と pos_hint を使用して設定すれば問題がないです**。pos_hint に関しては、特に説明をしないので興味のある方は実際にコードを書いてみて試してみてください。

hint の有効な値ですがマニュアルをみると 0.1〜1 が有効な範囲となっていますが、実際にコードを書くとわかるのですが、値を *10 や100 といれても有効に機能しています*ので、マニュアルか Kivy 内部の実装のどちらかがあっていないようです。

7. サイズの調整 2

実行結果



コード

（3. の内容と変更はありません。）

Kv ファイル

Kv Language は以下の通りです。

```
BoxLayout:
    orientation: 'vertical'     # 'horizontal'だと横一列

    Label:
        text: "Good Morning"
        size_hint_y: 0.5
    Label:
        text: "Hello"
        size_hint_y: 1
    Label:
```

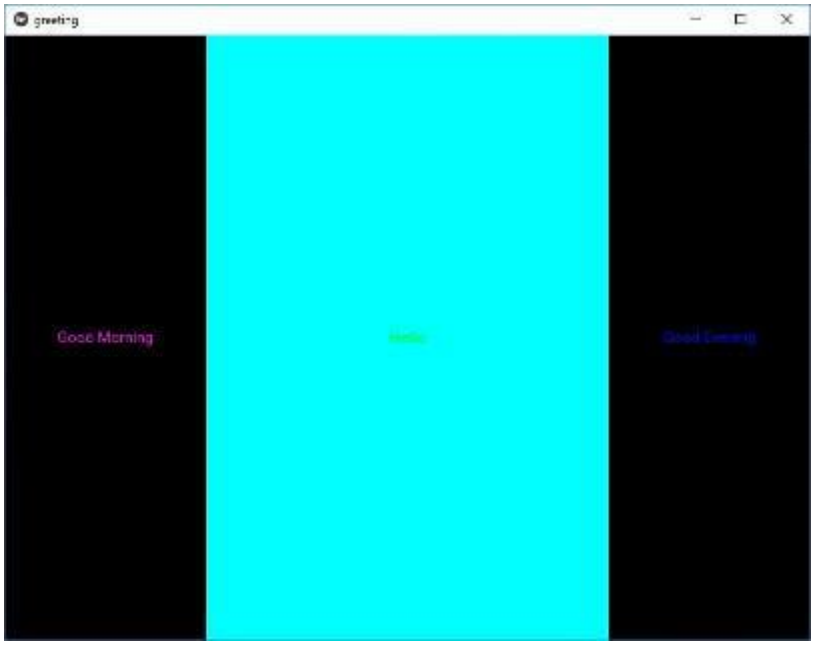
```
        text: "Good Evening"
        size_hint_y: 0.5
```

解説

6 の場合と値を変えてみました。

8. 文字の色とラベルの色の変更

実行結果



コード

```
#-*- coding: utf-8 -*-

from kivy.app import App


class TestApp(App):

    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = 'greeting'    # ウィンドウの名前を変更

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

Kv ファイル

```
BoxLayout:

    Label:
        text: "Good Morning"
        color: 1,0,1,1
        size_hint_x: 0.5
    Label:
        text: "Hello"
        color: 0,1,0,1 # rgba
        size_hint_x: 1

        canvas.before:
            Color:
                rgba: 0, 1, 1, 1
            Rectangle:
                pos: self.pos
                size: self.size
    Label:
        text: "Good Evening"
        color: 0,0,1,1 # rgba
        size_hint_x: 0.5
```

解説

Python 側では TestApp クラスの初期化の際に以下のコマンドを入れています。

```
self.title = 'greeting'    # ウィンドウの名前を変更
```

こうすることで起動時の window の左上に表示されるタイトル名を変更することができます。

参考

- タイトル変更（http://code.tiblab.net/python/kivy/app_title）

また、Kv ファイルですが、Label に「color」のパラメータを新たに追加しています。「color」は文字の色のパラメータですとる値は[r(赤色),g（黄色）,b（青色）,a（透明度）]のリスト構造です。有効な値の範囲は 0〜1 です。

Kivy では色を指定するのに他にも HSV モードなど色を指定するいくつかの方法がありますが、**色を表示するのは color を使う**ことを覚えていれば十分です。
また Label の背景の色を変更するですが、これは以下のコードで実現しています

```
        canvas.before:              #①
            Color:                      #②
                rgba: 0, 1, 1, 1
            Rectangle:                #③
                pos: self.pos
                size: self.size
```

① canvas コマンドで色をつけます。before はここでは詳しく説明しません、canvas に関しては Programming Guide(翻訳済み) » Graphics (翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/graphics.html）を読んでください。
②に関しては（R,G,B,A）＝（0,1,1,1）で色を付けます。Label と違うのは冒頭の C が大文字であることに注意してください。
③Rectangle は矩形（四角形）を表示するコマンドです。pos パラメータは表示する座標、size は描画する範囲の幅と高さを指定しています。ここで pos と size の値に self を使用していますが、**self は Kv の予約語でその Widget を取得するのに使用します**。今回は Hello を表示する Label 内ですので、self.pos は Label の xy 座標、self.size はその Label の幅と高さの値を指しています。
これにより、Hello を表示する Label の座標と幅と高さ分の(0.1.1.1.1)の色の四角形を描画できます。図形は四角形の他にも三角形、円、線を描くことができます。

参考

- canvas の説明（API Reference）（https://kivy.org/docs/api-kivy.graphics.instructions.html）

[Note] Kv の予約語について

Kivy Language にはあらかじめシステム上で使用が決められている予約語がいくつかあります。代表的なものは以下の物があります。

- self ・・・ その Widget を取得
- root ・・・ rootWidget を取得
- app ・・・ App を取得

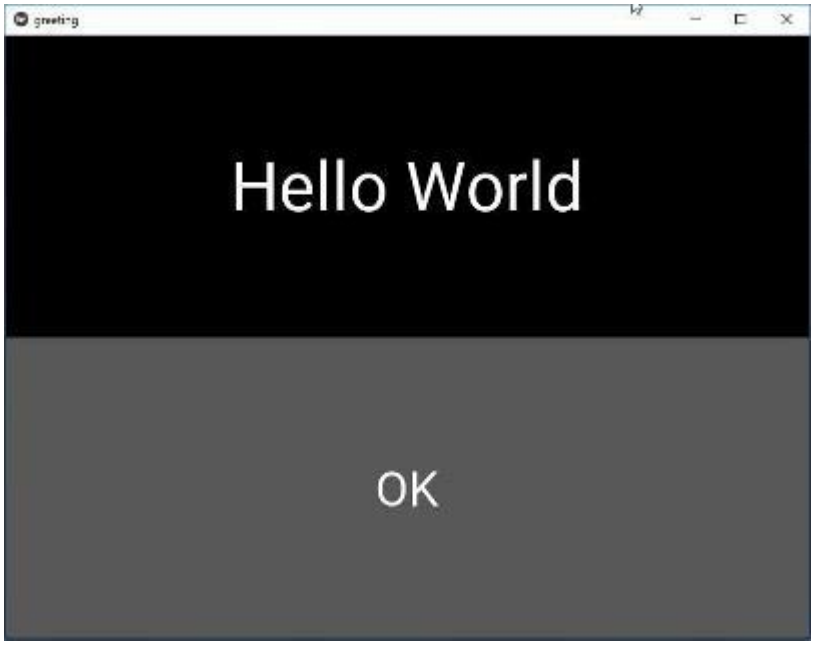
※ Kv ファイルの予約語（http://code.tiblab.net/python/kivy/kv_reserved_keywords）から引用

参考

- Kv ファイルの予約語（http://code.tiblab.net/python/kivy/kv_reserved_keywords）
- 予約語の説明（API Reference）（https://kivy.org/docs/api-kivy.lang.html#value-expressions-on-property-expressions-ids-and-reserved-keywords）

9. ボタンの追加

実行結果



コード

（8. の内容と変更はありません。）

Kv ファイル

Kv Language は以下の通りです。

```
TextWidget:


<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        # ラベル
        Label:
            id: label1
```

```
            font_size: 68
            text: "Hello World"

        Button:
            id: button1
            text: "OK"
            font_size: 48
```

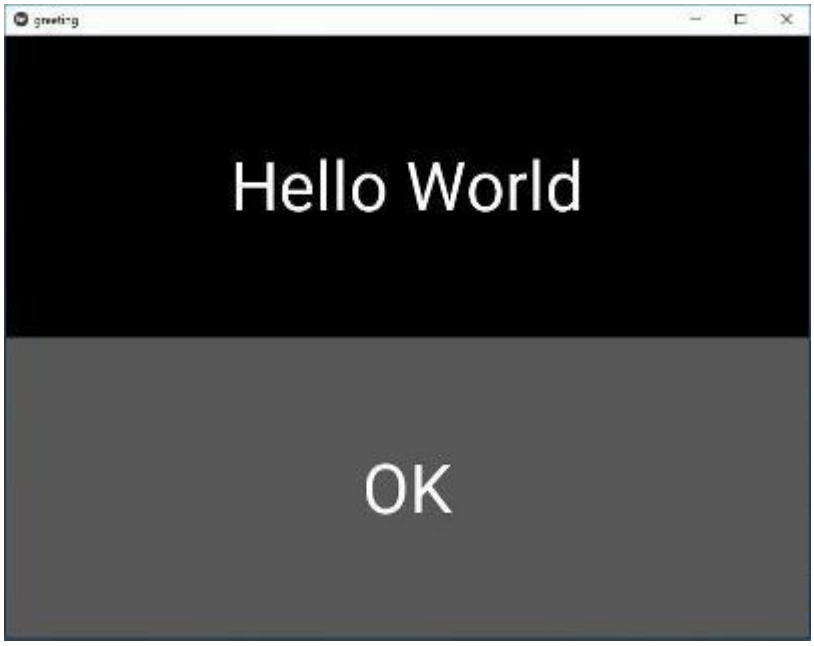
解説

新たに Button を追加しました。Button は Label のサブクラスなので Label で使用していたパラ―メータも使えます。今回のプログラムは起動して OK ボタンをクリックしてもクリック時にボタンの色が変わるだけで何も起きません。ここで出来たのはあくまでも Label と Button を使用したレイアウトができただけでボタンに対して何らかのアクション（Event）を起こした際の動作を登録しておりません。動作の登録に関しては次の項目で説明します。また font_size は表示される文字のサイズになります。デフォルトでは 15 になっていますが、それだと表示サイズが小さかったので大きくしました。

また widget ごとに新たに id というパラ―メータを追加していますが今回はふれません。参考の項目で使用方法を説明します。

10. ボタンを押した場合の機能の追加

実行結果



コード

コードは以下の通りです。

```
#-*- coding: utf-8 -*-

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty

class TextWidget(Widget):
    text = StringProperty()    # プロパティの追加

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        self.text = ''
```

```
    def buttonClicked(self):        # ボタンをクリック時
        self.text = 'Hello World'


class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = 'greeting'

    def build(self):
        return TextWidget()

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

Kv ファイル

Kv Language は以下の通りです。

```
TextWidget: # ルートに追加

<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        # ラベル
        Label:
            id: label1
            font_size: 68
            text: root.text    # root(TextWidget)の変数テキストの値を取得

        Button:
            id: button1
            text: "OK"
            font_size: 68
            on_press: root.buttonClicked()  # ボタンをクリックした時に python 側の関数を呼ぶ
```

解説

プログラムを実行すると、起動時にはラベルには何も表示されませんが、OK ボタンをクリックすると、「Hello World」の文字がラベルに表示されます。 まず python 側のコマンドですが、

```
from kivy.properties import StringProperty
```

として `kivy.properties` から `StringProperty` を import しています。 kivy の properties はボタンが押された場合などのイベントが実行された場合のオブジェクトの属性に値の変更があった場合に実行する仕組みの事です。今回はボタンが押されたときに、ラベルの文字が変化するのに使用します。プロパティですが、文字列や数字、辞書型ごとに型があります。型としては以下があります。

- StringProperty
- NumericProperty
- BoundedNumericProperty
- ObjectProperty
- DictProperty
- ListProperty
- OptionProperty
- AliasProperty
- BooleanProperty
- ReferenceListProperty

プロパティの詳細は Programming Guide(翻訳済み) » Events and Properties(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/events.html）を参考にしてください。 実際のコード上ではこの後、TextWidget クラスでクラス変数 text を宣言しておりそこで StringProperty() を代入しています。その後、「**init**」メソッドでは text には空白を代入しており、buttonClicked()関数を作成して、その中で text に「Hello World」の文字を代入しています。

```
class TextWidget(Widget):
    text = StringProperty()    # プロパティの追加

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        self.text = ''

    def buttonClicked(self):        # ボタンをクリック時
```

```
        self.text = 'Hello World
```

一方、kv ファイルですが、ラベルとボタンで以下のようになっています。

```
        # ラベル
        Label:
            id: label1
            font_size: 68
            text: root.text    # root(TextWidget)の変数テキストの値を取得

        Button:
            id: button1
            text: "OK"
            font_size: 68
            on_press: root.buttonClicked()  # ボタンをクリックした時に python 側の関数を呼ぶ
```

まず Label の text パラメータには、root.text が設定されています。root は Kv Language の予約語で root ウィジェットを指しています。ですのでここでは TextWidget のクラス変数 text を指しています。

また Button では on_press というパラ―メータが追加されています。on_press はあらかじめ Kivy で用意されているイベントで Button を押した際に実行されるイベントです。この場合は、 root.buttonClicked()を設定して、これにより OK のボタンが押されたときに TextWiget の buttonClicked()が実行されます。

Button のイベントにはこのほかには「on_release」がありこちらはボタンが離された場合に実行されるイベントになります。

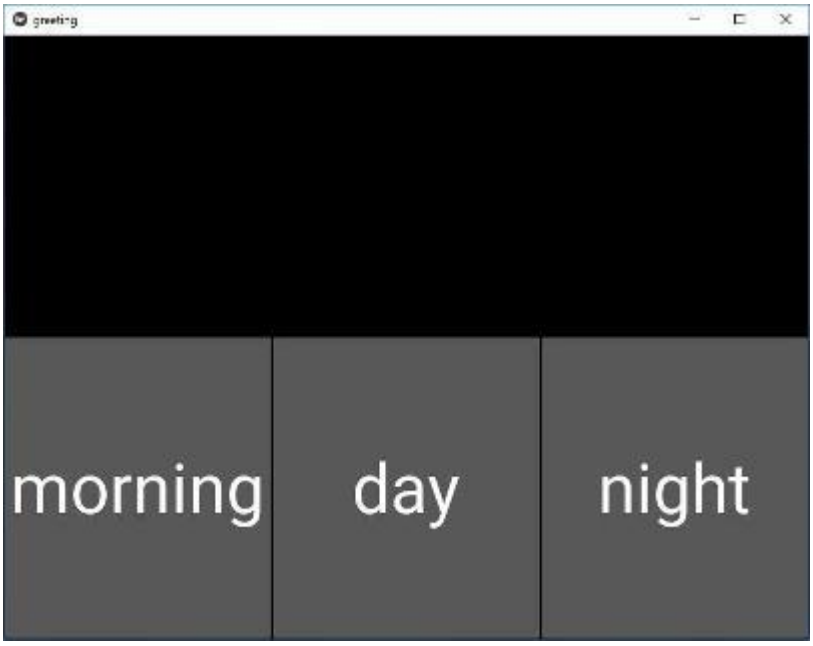
参考

- Button Behavior(API Reference)（https://kivy.org/docs/api-kivy.uix.behaviors.button.html#kivy.uix.behaviors.button.ButtonBehavior）

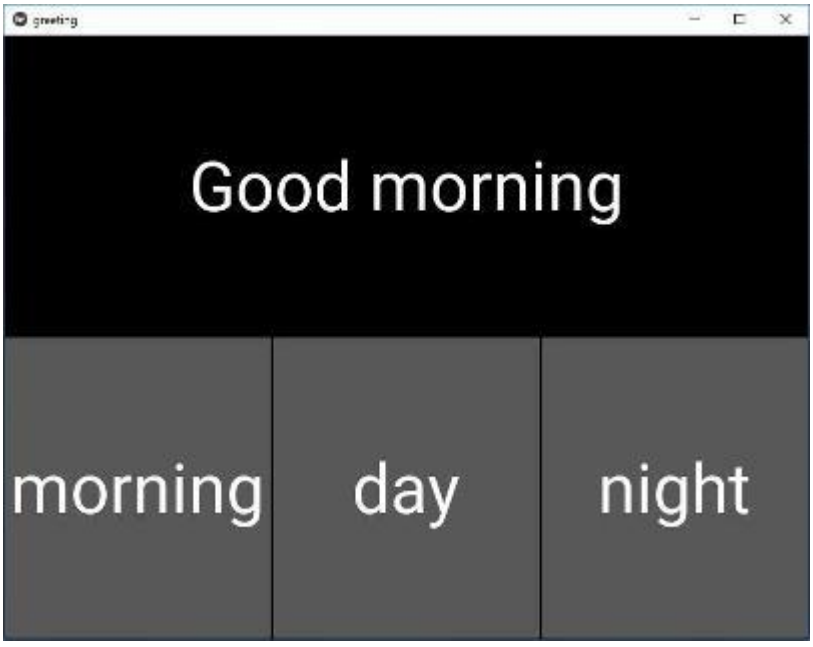
11. 複数のボタンを追加した場合

実行結果

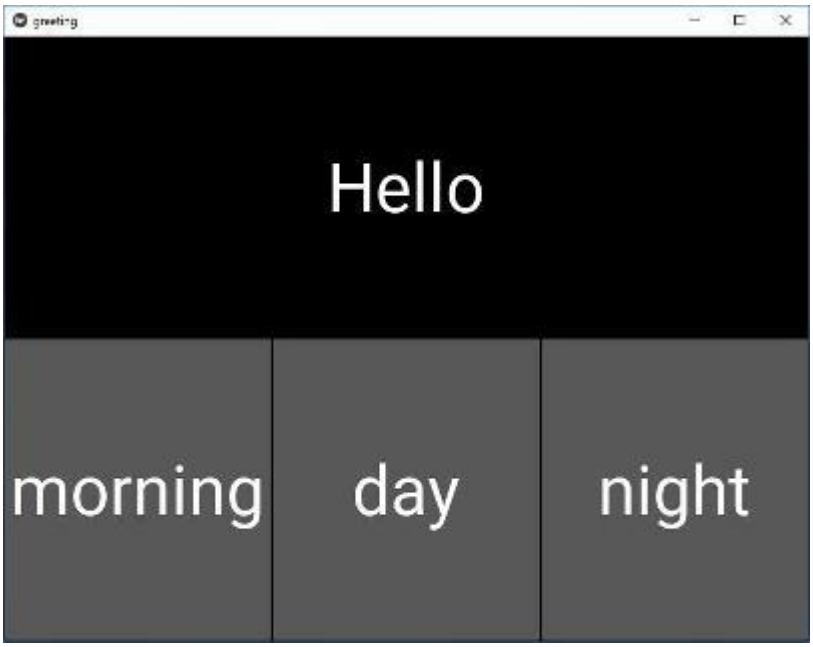
起動時



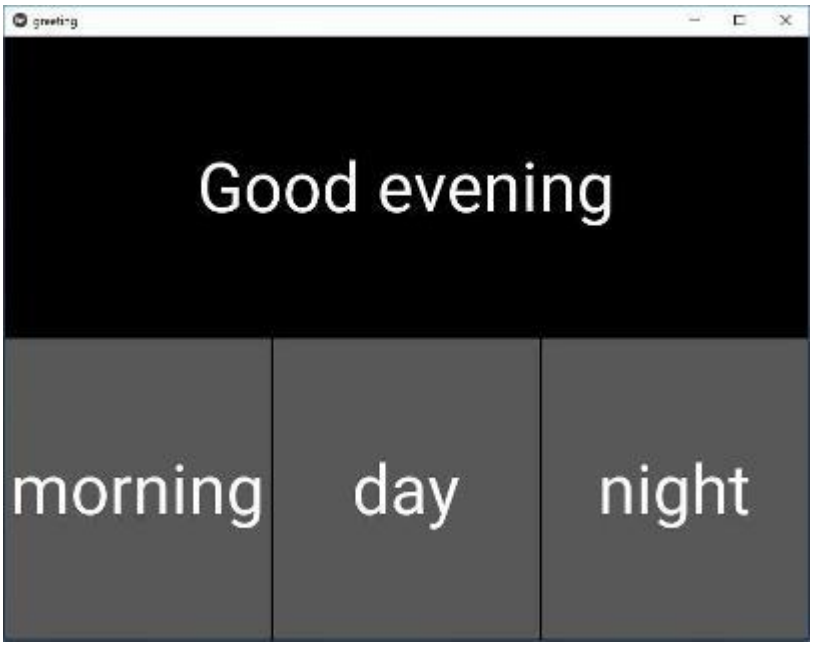
morining をクリック



day をクリック



night をクリック



コード

コードは以下の通りです。

```
#-*- coding: utf-8 -*-

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty

class TextWidget(Widget):
    text = StringProperty()

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        self.text = ''

    def buttonClicked(self):
        self.text = 'Good morning'

    def buttonClicked2(self):
        self.text = 'Hello'

    def buttonClicked3(self):
        self.text = 'Good evening'


class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = 'greeting'

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

Kv ファイル

Kv Language は以下の通りです。

```
#-*- coding: utf-8 -*-



TextWidget: # ルートに追加

<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        # ラベル
        Label:
            id: label1
            font_size: 68
            text: root.text

        BoxLayout:
            Button:
                id: button1
                text: "morning"
                font_size: 68
                on_press: root.buttonClicked()  # ボタンをクリックした時

            Button:
                id: button2
                text: "day"
                font_size: 68
                on_press: root.buttonClicked2()  # ボタンをクリックした時

            Button:
                id: button3
                text: "night"
                font_size: 68
                on_press: root.buttonClicked3()  # ボタンをクリックした時
```

解説

まず python ファイル側ですが、こちらは前の内容から複数ボタンが押された場合の関数が追加されただけなので説明を省略します。

Kv Language ですがボタンを押した際の動作は前回に説明をしたので問題ないかと思います。問題はレイアウトのほうで、今回は以下の構造をしています

```
<TextWidget>:
    BoxLayout:      #①
        orientation: 'vertical'
        size: root.size

        Label:
        ～省略～
        BoxLayout: #②
            Button:
      ～省略～
            Button:
      ～省略～
              Button:
         ～省略～
```

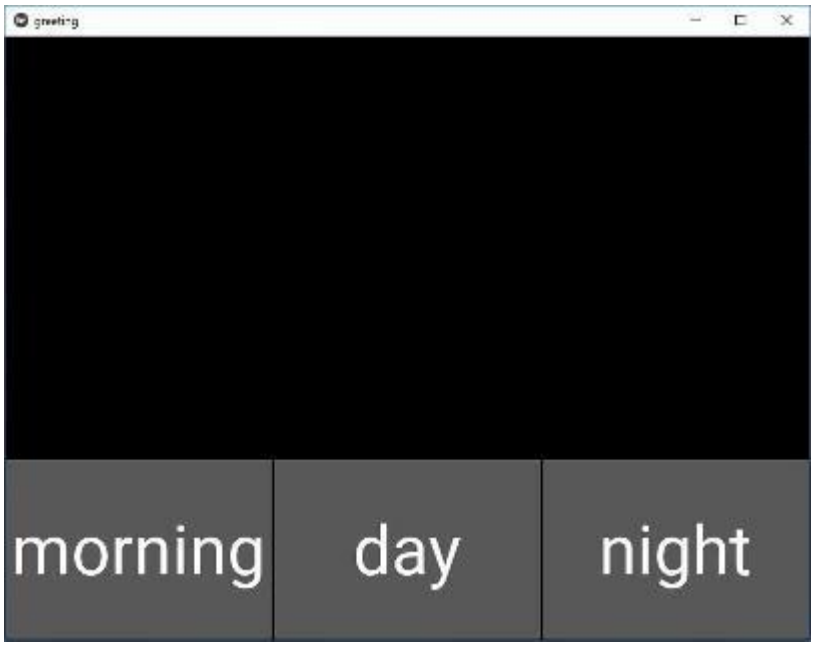
BoxLayout が 2 回使われていますが、まず①の BoxLayout で orientation の 設定が「vertical」になっています。これにより画面が上下に 2 分割されたレイアウトができます。その後、Label を追加して、上下分割された画面のうち上部に配置されます。問題は画面下の部分の方です。配置に際して②で BoxLayout が配置されます。orientation のパラ―メータの設定がないですが、orientation はデフォルトで「horizonal」に設定されています。これにより、画面下部は横並びのレイアウトになり、今回は Button が 3 つ追加されていますので、結果として横にボタンが 3 つ並べたレイアウトになります。
BoxLayout を 2 つ組み合わせることにより、画面の上半分 は 1 つ、下半分は水平に 3 分割のレイアウトができます。
Kivy では複数の Layout を組み合わせることで複雑なレイアウトを設置することが可能になります。

12. レイアウトを変更

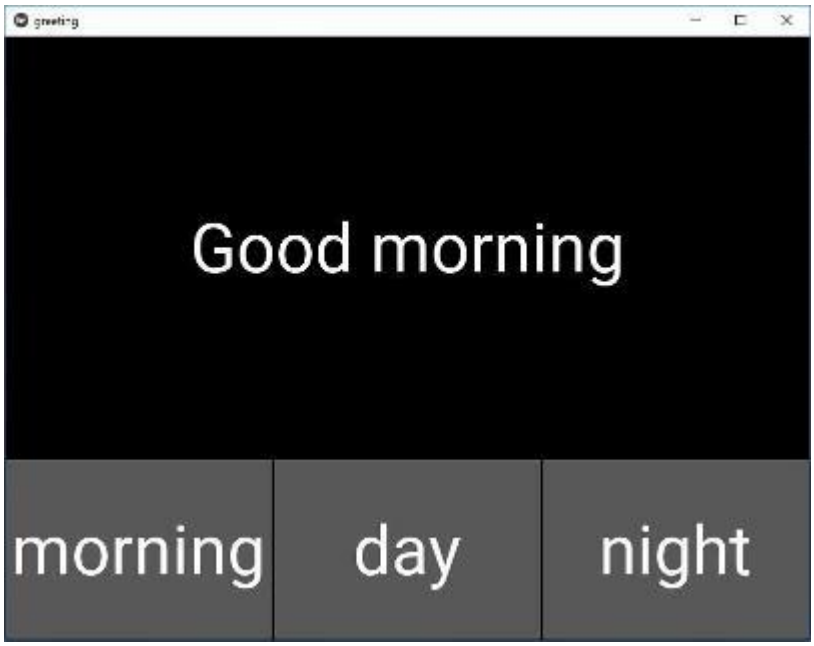
実行結果

実行結果は以下の通りです。

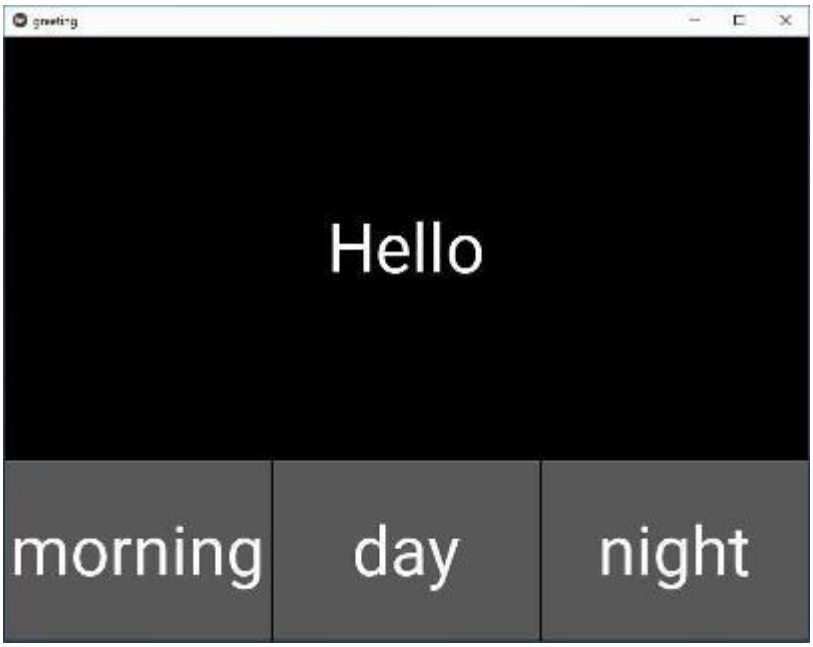
起動時



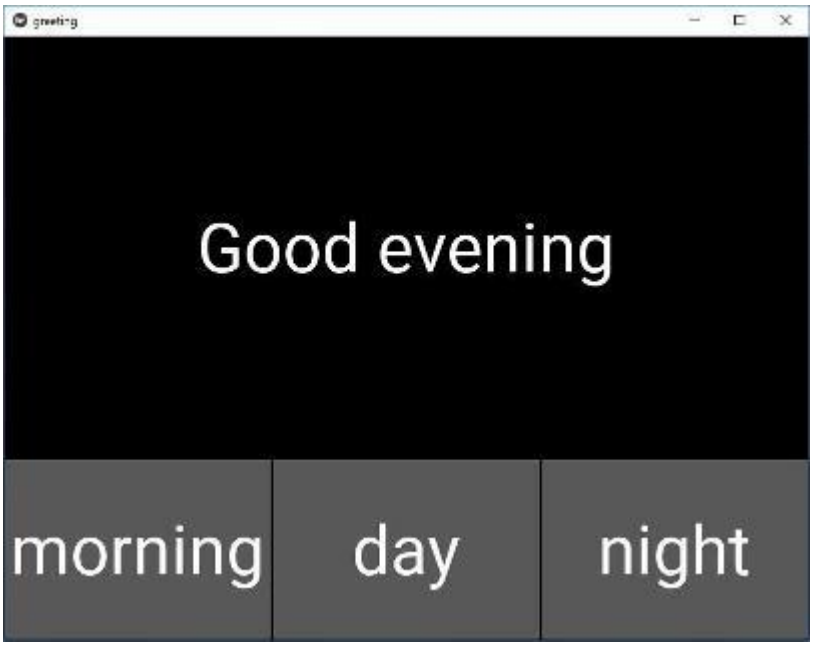
morining をクリック



day をクリック



night をクリック



コード

1. と変化ありません。

Kv ファイル

Kv Language は以下の通りです。

```
<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        # ラベル
        Label:
            size_hint_y: 0.7    # 画面全体の 70%を表示するように調整
            id: label1
            font_size: 68
            text: root.text

        BoxLayout:
            size_hint_y: 0.3     # 画面全体の 30%を表示するように調整
            Button:
                id: button1
                text: "morning"
                font_size: 68
                on_press: root.buttonClicked()  # ボタンをクリックした時

            Button:
                id: button2
                text: "day"
                font_size: 68
                on_press: root.buttonClicked2()  # ボタンをクリックした時

            Button:
                id: button3
                text: "night"
                font_size: 68
                on_press: root.buttonClicked3()  # ボタンをクリックした時
```

解説

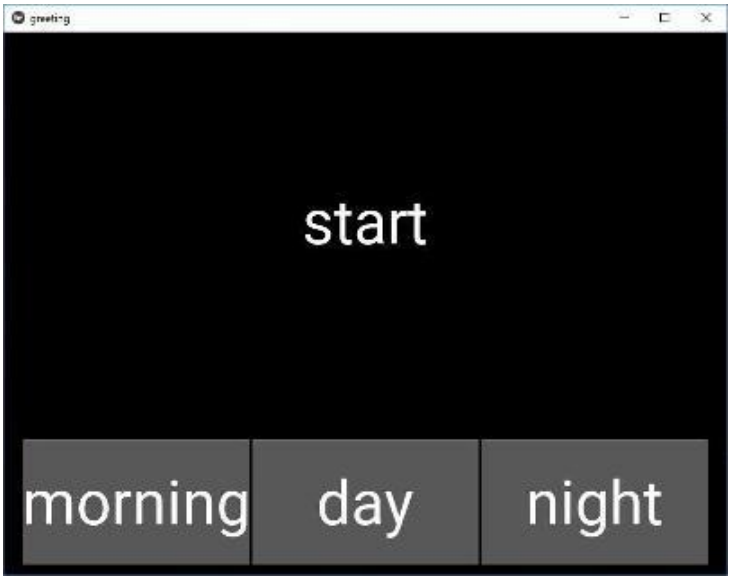
Kv ファイル側で size_hint_y を使用してサイズ調整をしています。上半分の、ラベルの縦のサイズが大きくなり、下半分のボタンのサイズが小さくなります。

13. クリック時の機能追加(文字の色を変更する)

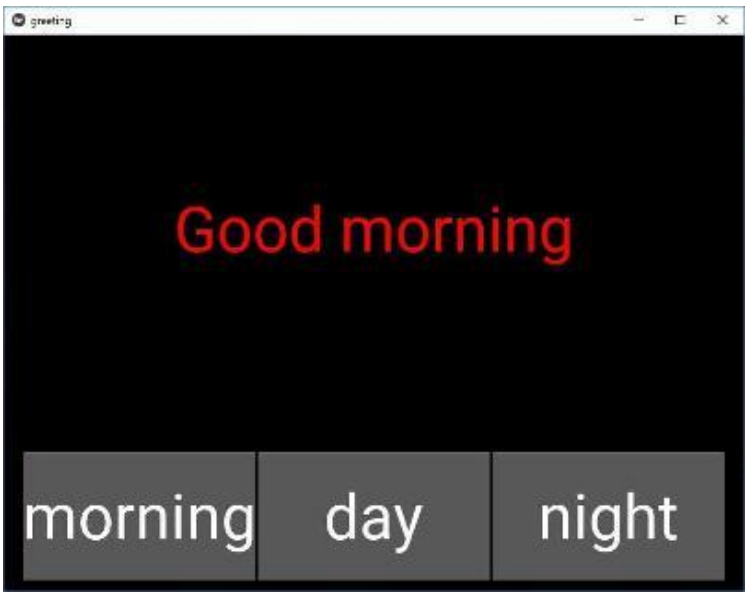
実行結果

実行結果は以下の通りです。

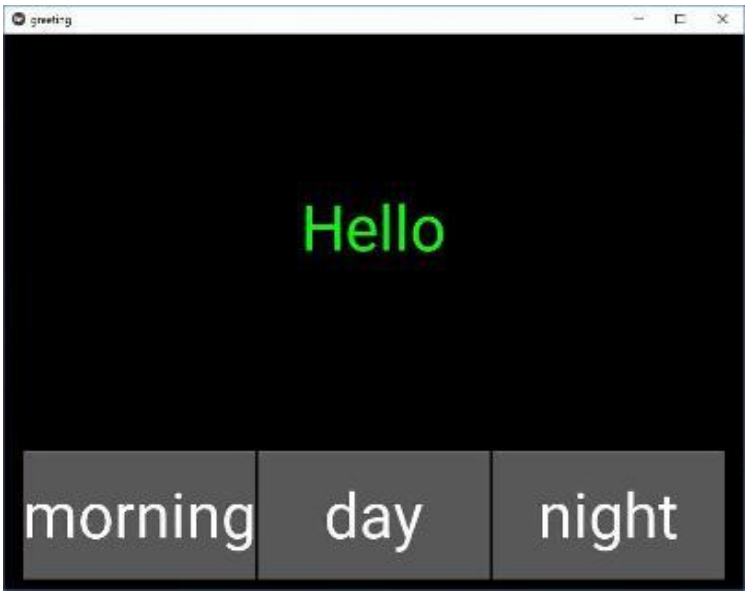
起動時



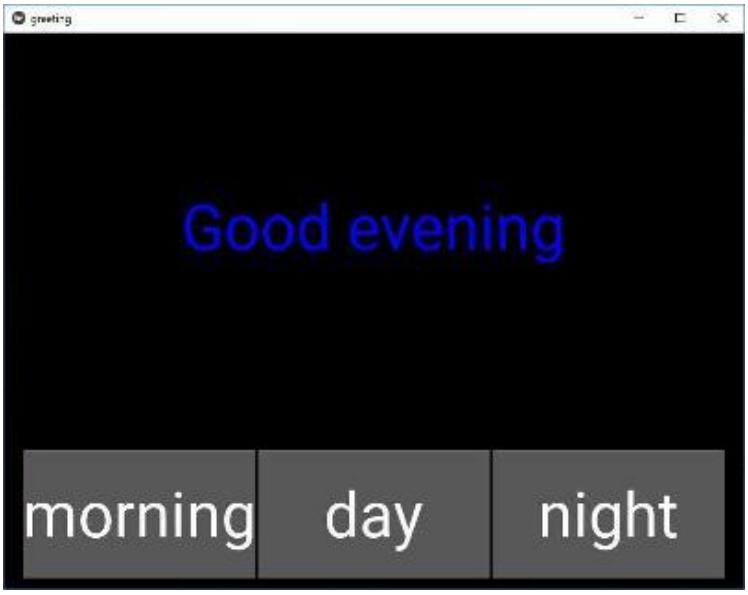
morining をクリック



day をクリック



night をクリック



コード

コードは以下の通りです。

```
#-*- coding: utf-8 -*-

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty, ListProperty

class TextWidget(Widget):
    text  = StringProperty()
    color = ListProperty([1,1,1,1])

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        self.text = 'start'

    def buttonClicked(self):
        self.text = 'Good morning'
        self.color = [1, 0, 0 , 1]

    def buttonClicked2(self):
        self.text = 'Hello'
        self.color = [0, 1, 0 , 1 ]

    def buttonClicked3(self):
        self.text = 'Good evening'
        self.color = [0, 0, 1 , 1 ]

class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = 'greeting'

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

Kv ファイル

Kv Language は以下の通りです。

```
#-*- coding: utf-8 -*-



TextWidget: # ルートに追加

<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        # ラベル
        Label:
            size_hint_y: 0.7
            id: label1
            font_size: 68
            text: root.text
            color: root.color

        BoxLayout:
            size_hint_y: 0.3
            padding: 20,30,20, 10
            Button:
                id: button1
                text: "morning"
                font_size: 68
                on_press: root.buttonClicked()  # ボタンをクリックした時

            Button:
                id: button2
                text: "day"
                font_size: 68
                on_press: root.buttonClicked2()  # ボタンをクリックした時

            Button:
```

```
                id: button3
                text: "night"
                font_size: 68
                on_press: root.buttonClicked3()  # ボタンをクリックした時
```

解説

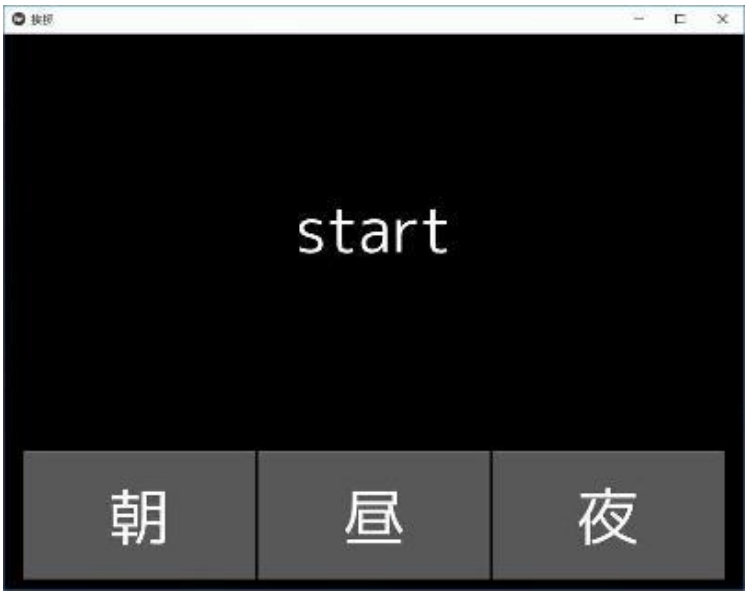
前回までのコードでラベルの表示する文字の色を変える機能を追加しています。
内容としては Python 側の TextWidget クラスでクラス変数を color を新たに ListProperty で宣言して値を実現しています。color は rgba の複数の値を格納するために ListProperty を使用しています。

14. 日本語の表示（フォントの追加）

実行結果

実行結果は以下の通りです。

起動時



朝をクリック

昼をクリック

夜をクリック

コード

コードは以下の通りです。

```
#-*- coding: utf-8 -*-

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty, ListProperty

from kivy.core.text import LabelBase, DEFAULT_FONT
from kivy.resources import resource_add_path

# デフォルトに使用するフォントを変更する
resource_add_path('./fonts')
LabelBase.register(DEFAULT_FONT, 'mplus-2c-regular.ttf') #日本語が使用できるように
日本語フォントを指定する


class TextWidget(Widget):
    text  = StringProperty()
    color = ListProperty([1,1,1,1])

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        self.text = 'start'

    def buttonClicked(self):
        self.text = 'おはよう'
        self.color = [1, 0, 0 , 1]

    def buttonClicked2(self):
        self.text = 'こんにちは'
        self.color = [0, 1, 0 , 1 ]

    def buttonClicked3(self):
        self.text = 'こんばんは'
```

```
        self.color = [0, 0, 1 , 1 ]

class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = '挨拶'

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

Kv ファイル

Kv Language は以下の通りです。

```
TextWidget:

<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        # ラベル
        Label:
            size_hint_y: 0.7
            id: label1
            font_size: 68
            text: root.text
            color: root.color

        BoxLayout:
            size_hint_y: 0.3
            padding: 20,30,20, 10
            Button:
                id: button1
                text: "朝"
                font_size: 68
                on_press: root.buttonClicked()
```

```
        Button:
            id: button2
            text: "昼"
            font_size: 68
            on_press: root.buttonClicked2()

        Button:
            id: button3
            text: "夜"
            font_size: 68
            on_press: root.buttonClicked3()
```

解説

Kivy ですが、windows では **Python34¥Lib¥site-packages¥kivy¥data¥fonts** 配下にフォントデータはありますが、デフォルトの文字コードは **Roboto で、Roboto は欧文フォントなのでそのままでは日本語は正しく表示されません**。
なので日本語を表示するためには、自分で日本語を表示するフォントを使用する必要があります。Python コード側で

```
from kivy.core.text import LabelBase, DEFAULT_FONT
from kivy.resources import resource_add_path

# デフォルトに使用するフォントを変更する
resource_add_path('./fonts')
LabelBase.register(DEFAULT_FONT, 'mplus-2c-regular.ttf') #日本語が使用できるように日本語フォントを指定する
```

としていますが、resource_add_path は kivy 内で読み込みパスを指定しています。
次に LabelBase.register で標準に使用するフォントで日本語が表示できるフォントを指定しています。今回は python のファイルと同じ階層に「fonts」フォルダを作成しそこに「mplus-2c-regular.ttf」ファイルを配置しています。その「mplus-2c-regular.ttf」ファイルを文字表示する際のフォントに指定しています。

日本語の表示するには大まかに分けて以下の 2 つの方法があるかと思います。

- OS ごとのシステムフォントで日本語が表示可能なフォントを指定する。
- フォントを用意して起動時にデフォルトフォントを用意する。

OS ごとのシステムフォントで日本語が表示可能なフォントを指定する方法ですが、kivy には platform という機能があり、これを使用すると実行時の OS を取得できます。これを用いて OS ごとにデフォルトで表示するフォントを変更することができます。日本でも実際にやられている方がいます。

参考

- Tofu issue 日本語フォントの表示 - Kivy Advent Calendar 2013（http://d.hatena.ne.jp/cheeseshop/20131202/1385936070）

ただし、このやり方だと未知の OS や Android などの機種ごとのカスタマイズが多い機種だと指定したフォントが存在しない場合があります。

フォントを用意して起動時にデフォルトフォントを用意する方法はフォントファイル（.ttf）を用意して、プログラムと同封して起動時にそのフォントを使用するように設定します。今回はこのやり方を使用しています。フォントは M+（http://mplus-fonts.osdn.jp/）というライセンスフリーなフォントを使用しています。
フォントの指定にはいろいろなやり方があり、いくつか紹介されていますので興味がある方は以下の箇所をのぞいてみてください。

参考

- Kivy アプリに日本語を表示させる（http://supportdoc.net/support-kivy/03jp.html）
- kivy で日本語表示（http://yukirinmk2.hatenablog.com/entry/2014/06/02/220608）
- Kv ファイルで日本語を使う（http://code.tiblab.net/python/kivy/kv_japanese）
- Kivy recipe: connecting font file（http://cheparev.com/kivy-connecting-font/）

Note Kv ファイルの文字コードについて

kv ファイルは Windows では Python3 系では「shift-jis」で 2 系の場合は「utf-8」で保存しないと正しく表示されません。

Note App の title について

Python2 系だと title を「u'日本語'」で入力してもエラーになります。以下のように main 側も修正することで実行が可能になります

```
import sys


if __name__ == '__main__':
```

```
    reload(sys)
    sys.setdefaultencoding('utf-8')
    Test3App().run()
```

15. 起動時の解像度を変更およびパラメータの使いまわしについて

実行結果

実行結果は以下の通りです。

起動時

コードは以下の通りです。

```
#-*- coding: utf-8 -*-
from kivy.config import Config
Config.set('graphics', 'width', '640')
Config.set('graphics', 'height', '480')

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty, ListProperty

from kivy.core.text import LabelBase, DEFAULT_FONT
from kivy.resources import resource_add_path
```

```
# デフォルトに使用するフォントを変更する
resource_add_path('./fonts')
LabelBase.register(DEFAULT_FONT, 'mplus-2c-regular.ttf') #日本語が使用できるように
日本語フォントを指定する


class TextWidget(Widget):
    text  = StringProperty()
    color = ListProperty([1,1,1,1])

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        self.text = 'start'

    def buttonClicked(self):
        self.text = 'おはよう'
        self.color = [1, 0, 0 , 1]

    def buttonClicked2(self):
        self.text = 'こんにちは'
        self.color = [0, 1, 0 , 1 ]

    def buttonClicked3(self):
        self.text = 'こんばんは'
        self.color = [0, 0, 1 , 1 ]

class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = '挨拶'

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

Kv Language は以下の通りです。

```
TextWidget:



<MyButton@Button>:
    font_size: 68

<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        # ラベル
        Label:
            size_hint_y: 0.7
            id: label1
            font_size: 68
            text: root.text
            color: root.color

        BoxLayout:
            size_hint_y: 0.3
            padding: 20,30,20, 10
            MyButton:
                id: button1
                text: "朝"
                #font_size: 68
                on_press: root.buttonClicked()

            MyButton:
                id: button2
                text: "昼"
                #font_size: 68
                on_press: root.buttonClicked2()

            MyButton:
                id: button3
                text: "夜"
```

```
                #font_size: 68
                on_press: root.buttonClicked3()
```

解説

画像だとわかりにくいですが、起動すると画面サイズが小さくなっているのがわかります。

```
from kivy.config import Config
Config.set('graphics', 'width', '640')
Config.set('graphics', 'height', '480')
```

Python ファイルがわの冒頭で以下のコマンドが追加されています。
Kivy はインストール直後のデフォルトの設定では、解像度が 800×600 で起動するようになっています。Programming Guide(翻訳済み) » Configure Kivy(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/config.html）に書いてありますが、Kivy には「config.ini」というファイルで基本的な設定はそこに記載されています。
Windows では *C:¥Users¥<ユーザー名.kivy* 配下にあります。「config.ini」の中身は以下のようになっています。

```
[kivy]
keyboard_repeat_delay = 300
keyboard_repeat_rate = 30
log_dir = logs
log_enable = 1
log_level = info
log_name = kivy_%y-%m-%d_%_.txt
window_icon =
keyboard_mode =
keyboard_layout = qwerty
desktop = 1
exit_on_escape = 1
pause_on_minimize = 0
config_version = 14

[graphics]
display = -1
fullscreen = 0
height = 600
left = 0
maxfps = 60
```

```
multisamples = 2
position = auto
rotation = 0
show_cursor = 1
top = 0
width = 800
resizable = 1
borderless = 0
window_state = visible
minimum_width = 0
minimum_height = 0

[input]
mouse = mouse
wm_touch = wm_touch
wm_pen = wm_pen

[postproc]
double_tap_distance = 20
double_tap_time = 250
ignore = []
jitter_distance = 0
jitter_ignore_devices = mouse,mactouch,
retain_distance = 50
retain_time = 0
triple_tap_distance = 20
triple_tap_time = 375

[widgets]
scroll_timeout = 250
scroll_distance = 20
scroll_friction = 1.
scroll_stoptime = 300
scroll_moves = 5

[modules]
```

このうち「graphics」の項目の height と width の値が起動時の解像度になります。
この config.ini の中身を書き換えて変更することも可能ですが、それだと配布した際にユー

ザーごとの config.ini の設定に依存することになりますので、このように一時的に変更することも可能です。

参考 Kivy で簡単なアプリを作ってみる
（http://tkitao.hatenablog.com/entry/2014/10/10/083252）

またこのやり方とは別に、windw()を使用して解像度を変更させる方法もあります。

```
from kivy.core.window import Window
#Window.size = (450, 600)
```

ただしこちらのやり方で Android などのモバイルのデバイスで表示させると解像度がおかしくなるので、モバイルで表示させる場合は window()を使用しないか、以前に説明した pratform を用いて OS の種類によっては表示させないなどの処理が必要です。

また Kv ファイルですが以下のようになっています

```
<MyButton@Button>:  # ①
    font_size: 68

<TextWidget>:
      ～省略～
            MyButton:  #②
                id: button1
                text: "朝"
                #font_size: 68
                on_press: root.buttonClicked()
            MyButton:
                id: button2
                text: "昼"
                #font_size: 68
                on_press: root.buttonClicked2()

            MyButton:
                id: button3
                text: "夜"
                #font_size: 68
                on_press: root.buttonClicked3()
```

前回との違いは、 ①で「MyButton@Button」という widget が宣言されているのと、②で前回までは「Button」が設定されていたところが「MyButton」に設定されています。kivy では widget 宣言時に **widget@widget の種類**と宣言することで@より後の widget の種類を継承することができます。今回は「Button」widget を宣言することで「Button」widget を継

承した「MyButton」widget を作成しました。MyButton のなかで font_size を設定することで、前回まで各 Button で font_size を個別に指定する必要がなくなり同一の値が使用できるようになります。 詳しくは Programming Guide(翻訳済み) » Kv language(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/lang.html）を読んでみてください。

16. 画像の表示

実行結果

実行結果は以下の通りです。

起動時

コード

コードは以下の通りです。

```
#-*- coding: utf-8 -*-
from kivy.config import Config
Config.set('graphics', 'width', '640')
Config.set('graphics', 'height', '480')

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty, ListProperty

from kivy.core.text import LabelBase, DEFAULT_FONT
from kivy.resources import resource_add_path

# デフォルトに使用するフォントを変更する
```

```
resource_add_path('./fonts')
LabelBase.register(DEFAULT_FONT, 'mplus-2c-regular.ttf') #日本語が使用できるように
日本語フォントを指定する



class TextWidget(Widget):
    text  = StringProperty()

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        pass

    def buttonClicked(self):
        pass

    def buttonClicked2(self):
        pass

    def buttonClicked3(self):
        pass

class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = '画像表示'

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

Kv Language は以下の通りです。

```
TextWidget:

<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        Image:
            source: './image/sample2.jpg'



        BoxLayout:
            size_hint_y: 0.3
            padding: 20,30,20, 10
            Button:
                id: button1
                text: "サンプル 1"
                font_size: 30
                on_press: root.buttonClicked()

            Button:
                id: button2
                text: "サンプル 2"
                font_size: 30
                on_press: root.buttonClicked2()

            Button:
                id: button3
                text: "サンプル 3"
                font_size: 30
                on_press: root.buttonClicked3()
```

解説

Kivy では画像を表示することが可能です。

```
        Image:
            source: './image/sample2.jpg'
```

と指定することで画像を表示できます。また size パラメータを使用することで表示サイズを変更したりすることも可能です。今回はボタンをクリックしても画像は切り替わりません。ボタンクリックによる画像切り替えは次の項目で説明します。

参考

- Image(API Reference)（https://kivy.org/docs/api-kivy.uix.image.html）

17. 画像の表示（ボタンによる画像切り替え）

実行結果

実行結果は以下の通りです。

起動時　or　サンプル 1 をクリック時

サンプル 2 をクリック時

サンプル 3 をクリック時

コード

コードは以下の通りです。

```
#-*- coding: utf-8 -*-
from kivy.config import Config
Config.set('graphics', 'width', '640')
Config.set('graphics', 'height', '480')

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty

from kivy.core.text import LabelBase, DEFAULT_FONT
from kivy.resources import resource_add_path

# デフォルトに使用するフォントを変更する
resource_add_path('./fonts')
LabelBase.register(DEFAULT_FONT, 'mplus-2c-regular.ttf') #日本語が使用できるように
日本語フォントを指定する
```

```
resource_add_path('./image')


class ImageWidget(Widget):
    source = StringProperty(None)

    def __init__(self, **kwargs):
        super(ImageWidget, self).__init__(**kwargs)
        pass

    def buttonClicked(self):
        self.source= './image/sample.jpg'

    def buttonClicked2(self):
        self.source = 'sample2.jpg'

    def buttonClicked3(self):
        self.source = 'sample3.jpg'


class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = '画像表示'

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

Kv ファイル

Kv Language は以下の通りです。

```
ImageWidget:

<ImageWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size

        Image:
            source: root.source



        BoxLayout:
            size_hint_y: 0.3
            padding: 20,30,20, 10
            Button:
                id: button1
                text: "サンプル 1"
                font_size: 30
                on_press: root.buttonClicked()

            Button:
                id: button2
                text: "サンプル 2"
                font_size: 30
                on_press: root.buttonClicked2()

            Button:
                id: button3
                text: "サンプル 3"
                font_size: 30
                on_press: root.buttonClicked3()
```

解説

説明ですが、前回までの内容を読まれると大体内容がわかるかと思いますので詳しい説明は省きますが、Python 側で StringProperty を用いてクラス変数 source を作成して、ボタンクリック時にそこに画像ファイルのパスを切り替えて表示しています。

今回、画像は加工済みの画像を表示していますが、buttonClicked()の処理で Pillow や OpenCV などの画像ライブラリ—を用いてフィルタ—をかけて画像ファイルを生成し、その画像ファイルを生成するといった処理ができるかと思います。
なお、画像ファイルを生成せずにバイト列にして直接読み込むことも可能です。
その際は Texture の blit_buffer()（https://kivy.org/docs/api-kivy.graphics.texture.html#kivy.graphics.texture.Texture.blit_buffer）を使用することで実現できます。

参考

- [Python] QR コード作成 - Kivy Advent Calendar 2013
  （http://d.hatena.ne.jp/cheeseshop/20131203）

まとめ

ここまでのことで、レイアウトの基本的な説明とボタンをクリックした際の動作の実行を説明しましたので簡単なアプリの実装はできるようになったかと思います。
ここまでの内容を読んで、公式マニュアルの翻訳サイト（https://pyky.github.io/kivy-doc-ja/）の「Programming Guide」を読み直すと色々な発見があると思いますので、一度通して読み直すことをお勧めします。

また Kivy のサンプルプログラムですが、Kivy の example 配下（https://github.com/kivy/kivy/tree/master/examples）には様々なサンプルがあるのでぜひ一度見てみるとよいと思います。
どういったプログラムがあるかは、Gallery of Examples(翻訳済み) » Gallery(翻訳済み)（https://pyky.github.io/kivy-doc-ja/examples/gallery.html）と Getting Started(翻訳済み) » Examples(翻訳済み)（https://pyky.github.io/kivy-doc-ja/gettingstarted/examples.html）に内容の説明があります。

Kivy 自体は機能がたくさんありますがこれを読んで何か参考になれば幸いです。
また有志による公式マニュアルの翻訳（https://github.com/pyKy/kivy-doc-ja）もやっておりますので興味があれば参加してみてください。2017 年は API を中心にやっていいこうと考えています。

参考　ファイルを選択する

Kivy でファイルを選択する際は FileChooser（https://kivy.org/docs/api-kivy.uix.filechooser.html）を使用してファイル選択ダイアログを開くことができます。

実際の使用法ですが以下のサイトで、他の方が実際の使用法をコードとともに公開しているので参考にしてみてください。

参考

- Kivy で、シンプルなミュージックプレイヤー①（https://torina.top/main/302/）

参考　日本語の入力について（注意）

以下はテキストボックスに文字を入れて「OK」ボタンをクリックするとラベルに入力された文字が表示されます。

実行結果

実行結果は以下の通りです。

コード

コードは以下の通りです。

```
#-*- coding: utf-8 -*-

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty

from kivy.core.text import LabelBase, DEFAULT_FONT
from kivy.resources import resource_add_path
```

```
# デフォルトに使用するフォントを変更する
resource_add_path('./fonts')
LabelBase.register(DEFAULT_FONT, 'mplus-2c-regular.ttf') #日本語が使用できるように
日本語フォントを指定する



class TextWidget(Widget):
    text = StringProperty()

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        self.text = ''

    def buttonClicked(self):
        self.text =  self.ids["text_box"].text



class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)

    def build(self):
        return TextWidget()

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

Kv ファイル

Kv Language は以下の通りです。

```
TextWidget: # ルートに追加

<TextWidget>:
    BoxLayout:
        orientation: 'vertical'
        size: root.size
        # ラベル
```

```
        Label:
            id: label1
            font_size: 68
            text: root.text

        BoxLayout:
            TextInput:
                id: text_box
                size_hint_x: 70
                font_size: 68
                focus: True
                on_text_validate: root.buttonClicked()
            Button:
                id: button1
                size_hint_x: 30
                text: "OK"
                font_size: 68
                on_press: root.buttonClicked()  # ボタンをクリックした時
```

解説

Textinput（https://kivy.org/docs/api-kivy.uix.textinput.html）を使用すると、テキストを入力できます。

ただし問題があります。日本語の入力自体はフォントを日本語用のフォントを指定すればできます。コピー＆ペーストで貼り付けもできます。問題は **OS によっては IME が開かないので文字が確定するまで全角文字が表示されない**という問題があります。公式でも問題（https://github.com/kivy/kivy/issues/434）になっていますが、問題の経緯として元々Kivy は文字入力を pygame というライブラリに依存しており、pygame の既知の問題で IME が表示されないという問題がありました。その後、Kivy 自体は pygame の依存をやめて SDL ベースで色々な機能を書き直して、pygame を使用しないようにしてきましたが、入力周りはそのあたりの修正ができていないようです。なので日本語入力を積極的に使用したい際は注意が必要です。現在のところ OS ごとの IME に関してはこんな感じです。

- windows:不可、IME は開かない
- MacOS:不可、IME は開かない
- Linux：可（？）、Ubuntu ではできるそうです。（未検証）
- Android:可、Nexus5 で試したところ日本語入力は問題なくできます。
- iOS:不可、IME は開かないそうです。(未検証)

ボタンをクリックした際のテキストの入力ですがそれは次の参考で説明します。

参考　kv ファイルの値を Python 側で取得する方法

「日本語の入力について（注意）」についてのコードを見ると TextInput は以下のコードになっています。

```
<TextWidget>:
        BoxLayout:
            TextInput:
                id: text_box    # ★注目
```

ここで注目すべきは「id」に「text_box」と入力しています。 一方、Python 側のコードですがボタンを押した際に呼ばれる関数 buttonClicked()は以下のようになっています。

```
    def buttonClicked(self):
        self.text =  self.ids["text_box"].text
```

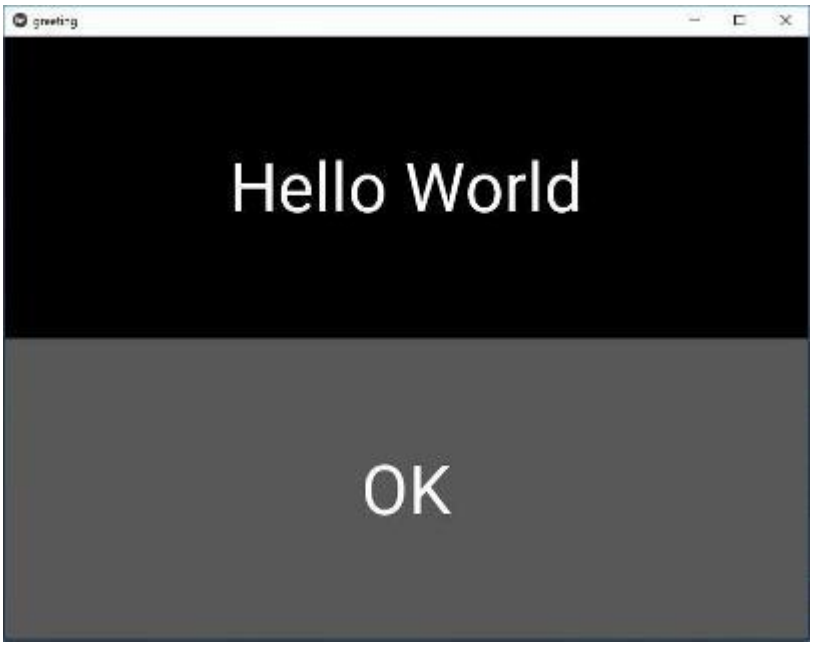
Kivy では id と場所のタグ付けがされている全てのウィジェットを **self.ids という辞書型のプロパティで一か所に管理しています**。
そうすることで id 名をキーにすることでそのウィジェットの各パラメータの値を取得できます。詳しくは Programming Guide(翻訳済み) » Kv language(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/lang.html）の「Python コードから Kv lang の内部で定義された widgets にアクセスする」の項目に記載されています。
ただし公式サイトのマニュアルですと、ObjectProperty を使用して値を習得する方がより良い方法とされています。ObjectProperty を使用して値を取得する方法は説明も手順が少し複雑なのと初心者向けではないのでここでは割愛します。

参考　Kivy 側で Python ファイルの関数の呼び方

ここまでで、Python 側のクラスのメソッド（関数）は、Kv ファイル側で、「self」や「root」といった予約語を使うことで利用することができました。
では逆に Python 側で書いたメソッドを呼びだして利用するにはどうしたらいいでしょうか。
方法としては Kv ファイル内で Python ファイルを import して利用する方法があります。

実行例

実行結果は以下の通りです。



コード

python ファイル(main.py)は以下の通りです。

```
#-*- coding: utf-8 -*-

from kivy.app import App
from kivy.uix.widget import Widget

from kivy.properties import StringProperty

def buttonClicked():    # この関数を呼び出します
```

```
    print("call def")
    text = 'Hello World'

    return text

class TextWidget(Widget):

    def __init__(self, **kwargs):
        super(TextWidget, self).__init__(**kwargs)
        self.text = ''

    def buttonClicked(self):
        self.text = 'Hello World'



class TestApp(App):
    def __init__(self, **kwargs):
        super(TestApp, self).__init__(**kwargs)
        self.title = 'greeting'

    def build(self):
        return TextWidget()

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

Kv Language は以下の通りです。

```
#: import main main
# ↑①main.py をインポートします。

TextWidget: # ルートに追加

<TextWidget>:
    BoxLayout:
        id: input
        orientation: 'vertical'
        size: root.size
        l_text:'aa'
```

```
        # ラベル
        Label:
            id: label1
            font_size: 68
            text: input.l_text  # aa からクリック後、'Hello World'に表示が変わります

        Button:
            id: button1
            text: "OK"
            font_size: 68
            on_press: input.l_text = main.buttonClicked()  # ②main.py の関数を呼び出
します。
```

解説

```
#: import main main
# ↑①main.py をインポートします。



<TextWidget>:
    BoxLayout:

   ～ 省略  ～
        Button:
            id: button1
            text: "OK"
            font_size: 68
            on_press: input.l_text = main.buttonClicked()  # ②main.py の関数を呼び出
します。
```

①で main.py を import しています。詳しい使用方法は以下の通りです

```
#: import <Kv ファイルで使いたい名前> <import したいライブラリ or ファイル名>
```

この書き方をすることで Kv ファイル側で特定のライブラリ、ファイル名を呼び出せます。
今回は①で main.py を main という名前で import した後に、②の OK ボタンをクリック時に main.py の buttonCliced()を呼び出しています。

参考

- Kivy Language -> Directives(API Reference)（https://kivy.org/docs/api-kivy.lang.html#lang-directives）

参考　GridLayout の使いかた

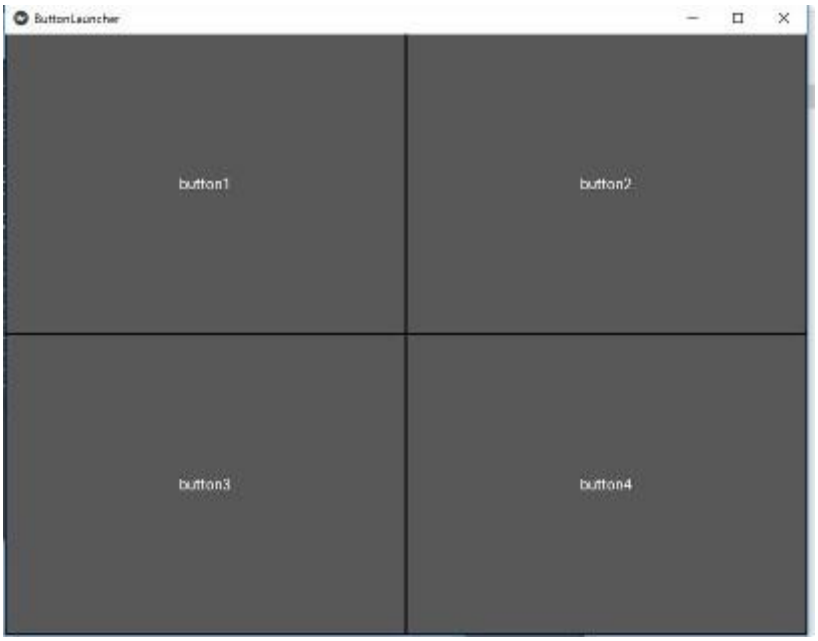
Kivy で GridLayout を使用しようとしてできなかったのという記事を見かけましたので、実際にコードを書いてみましたので載せます。

参考

- Python の GUI フレームワーク kivy を使ったけどムズすぎて断念した in Windows7 and Python2.7（http://d.hatena.ne.jp/shouh/20161015/1476489961）

実行例

実行結果は以下の通りです。



おそらく作者の方がやりたかったのはこのような感じかと思います。

コード

実際のコードですが、実装ですが 2 案用意してみました

案 1

python ファイルは以下の通りです。

```
# -*- coding: utf-8 -*
import kivy

from kivy.app import App
from kivy.config import Config
from kivy.uix.widget import Widget
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label

#from kivy.lang import Builder
#Builder.load_file('buttonlauncher.kv')

class MyWidget(GridLayout):
    def __init__(self):
        super(MyWidget, self).__init__()

class ButtonLauncherApp(App):
    def build(self):
        return MyWidget()



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

Kv Language（buttonLauncher.kv）は以下の通りです。

```
<MyWidget>:
    rows: 2
    cols: 2

    Button:
        text: 'button1'
    Button:
        text: 'button2'
    Button:
        text: 'button3'
```

```
    Button:
        text: 'button4'
```

案 2

python ファイルは以下の通りです。

```
# -*- coding: utf-8 -*-

import kivy
kivy.require('1.9.0')

from kivy.app import App
from kivy.config import Config
from kivy.uix.widget import Widget
from kivy.uix.label import Label

from kivy.lang import Builder
Builder.load_file('buttonlauncher2.kv')

class MyWidget(Widget):
    def __init__(self ,**kwargs):
        super(MyWidget, self).__init__()

class ButtonLauncher2App(App):
    def build(self):
        return MyWidget()



if __name__ == '__main__':
    ButtonLauncher2App().run()
```

Kv Language（buttonLauncher2.kv）は以下の通りです。

```
<MyWidget>
    GridLayout:
        rows: 2
        cols: 2
        size: root.size    # ★画面全体の値を取得する
```

```
        Button:
            text: 'button1'
        Button:
            text: 'button2'
        Button:
            text: 'button3'
        Button:
            text: 'button4'
```

解説

GridLayout 自体はちょっと使用に癖があるのと、おそらく BoxLayout 内で使用することを想定しているレイアウトのような気がしています。レイアウトに関しては BoxLayout をまず使用してみて、そこからレイアウトの組み立てをしてみるといいと思います。

参考　設定画面

Kivy では実行時に F1 キーを押すと、コンフィグ画面が表示されます。これにより、Log レベルを変更など、様々なことが可能です。

参考

- Kivy の設定画面について（http://qiita.com/Ds110/items/457db343467178957264）
- Application（API Reference）（https://kivy.org/docs/api-kivy.app.html)

参考　起動オプション

私も知りませんでしたが、Kivy では起動時にパラメータを与えるだけで、widget の階層構造を見ることができ、Property の値を確認できるモードがあります。参考のリンク先を一読することをお勧めします。

参考

- Kivy アプリ実行時に引数を与えるだけで使えるようになる便利なツール（http://qiita.com/gotta_dive_into_python/items/caa3e0f41ddfaedcdb95)
- Modules（API Reference）（https://kivy.org/docs/api-kivy.modules.html）

Kivy を使用して作成に向いているアプリ

起動時の画面からボタンを押して、2〜3 画面に遷移するアプリなどは向いています。頻繁に画面を遷移してかつ、値を保持するようなアプリには向いていないのではと個人的には思っています。

2章　～電卓を作成する～

あらまし

前の章の「Kv Language の基本」の使い方で、Kivy の基本的な使い方がなんとなくわかったかと思います。
今回は実際に簡単なアプリを作成してさらに理解を深めていきます。

作成するもの

作成するものは電卓アプリです。ボタンで数字と演算子を入力し「＝」ボタンをおすと演算結果が表示されます。また切り替えを押すとデザインが変わります

あらたに習得する内容

あらたに習得する内容は以下の通りです

- GridLayout の使い方
- ActionBar、およびその周りの使い方
- get_color_from_hex()による色の指定の仕方
- clear_widgets() と Factory(),add_widget()の使用方法

参考リンク

Kivy で電卓アプリを作成することは海外のチュートリアルではよく取り合げられています。

参考にしたものをあげます。

- Kivy Tutorial 3 : Kivy Calculator
  （https://www.youtube.com/watch?v=6yMHHC36tT0&feature=youtu.be）
- 電卓のソース 1（https://github.com/tapm/kivy_calculator）
- 電卓のソース 2（https://github.com/diegodukao/calkvlator）

検証環境

検証環境は以下の通りです。

OS: Windows10 64bit

Kivy:1.9.1

Python3.4※ ※Python2 系でもコードは実行できますが、日本語表示の箇所でエラーになります。その際は、コードを変更してみてください。（例:'日本語'→u'日本語'）
また Kv Language ファイルですが、Python3 系の場合は”Shift-jis”で Python2 系は"utf-8"で保存してください。

以下、実際にコードと結果を載せていきます。

実行結果

実行結果は以下の通りです。

起動すると画面が表示されます。通常の電卓と同じ動作が可能です。※1 頭のバーの切り替えボタンを押すと以下のデザインに切り替わります※2

※1「.」などの少数点表示など一部の機能は未実装です。
※2 あくまで切り替わるだけでボタンを押した際の動作などは未実装です。

コード

Python 側のプログラム(main.py)は以下の通りです。

```
# -*- coding: utf-8 -*
from kivy.config import Config
Config.set('graphics', 'width', '400')
Config.set('graphics', 'height', '800')
import kivy
kivy.require('1.9.1')

from kivy.uix.boxlayout import BoxLayout
from kivy.app import App
from kivy.core.text import LabelBase, DEFAULT_FONT
from kivy.core.window import Window
from kivy.properties import BooleanProperty
from kivy.utils import get_color_from_hex
from kivy.resources import resource_add_path

from kivy.factory import Factory

#from kivy.core.window import Window
#Window.size = (450, 600)

# デフォルトに使用するフォントを変更する
resource_add_path('fonts')
LabelBase.register(DEFAULT_FONT, 'mplus-2c-regular.ttf') #日本語が使用できるように
日本語フォントを指定する



class Calculator1(BoxLayout):

    clear_bool = BooleanProperty(False)

    def print_number(self, number):
        '''入力された値が入る'''
        if self.clear_bool:
            self.clear_display()
```

```
        text = "{}{}".format(self.display.text, number) # 今までの入力された文字列と
入力された値が表示される
        self.display.text = text

        print("数字「{0}」が押されました".format(number))

    def print_operator(self, operator):
        if self.clear_bool:
            self.clear_bool = False

        text = "{} {} ".format(self.display.text, operator)
        self.display.text = text

        print("演算子「{0}」が押されました".format(operator))

    def print_point(self, operator):
        # 「.」が押されれた場合の処理

        print("（未実装）演算子「{0}」が押されました".format(operator))

    def clear_display(self):
        self.display.text = ""
        self.clear_bool = False

        print("「c」が押されました")
    def del_char(self):
        ''' "<x"を押された場合の計算結果を表示  '''

        self.display.text = self.root.display.text[:-1]

        print("「<x」が押されました")

    def calculate(self):
        ''' "="を押された場合の計算結果を表示  '''
        try:
            self.display.text = str(eval(self.root.display.text)) # 単一の式を評価
例：eval("5 + 10")  は 15 になる
```

```
            self.clear_bool = True

            print('計算完了')
        except:
            # 数字を入力せずに  '='を押した場合などのエラー対策
            print('error  入力ミス')


#class Calculator2(BoxLayout):
#    def __init__(self, **kwargs):
#        super(Calculator2, self).__init__(**kwargs)




class CalculatorRoot(BoxLayout):
    def __init__(self, **kwargs):
        super(CalculatorRoot, self).__init__(**kwargs)

    def change_calc(self):
        self.clear_widgets()
        self.add_widget(Calculator1())

    def change_calc2(self):
        self.clear_widgets()
        calc2 = Factory.Calculator2()
        #calc2 = Calculator2()

        self.add_widget(calc2)



class CalculatorApp(App):
    def __init__(self, **kwargs):
        super(CalculatorApp, self).__init__(**kwargs)

        self.title = '電卓'
    pass
```

```
if __name__ == "__main__":
    Window.clearcolor = get_color_from_hex('#FFFFFF')
    CalculatorApp().run()
```

Kv ファイル

Kv ファイル(calculator.kv)は以下の通りです。

```
#: import get_color_from_hex kivy.utils.get_color_from_hex

CalculatorRoot




<CalculatorRoot>
    Calculator1
        id: calculator1

<calculator1>
# 画面全体のレイアウト
    orientation: "vertical" # オブジェクトを水平に配置
    display: display_input

    ActionBar:

        ActionView:
            ActionPrevious:
                title: '電卓'
                with_previous: False # true だと頭のロゴがボタンになる？

            ActionButton:
                text: '切り替え'
                on_press: print("push ");app.root.change_calc2()

#            ActionButton:
#                text: '音'
```

```
#               icon: 'atlas://data/images/defaulttheme/audio-volume-high'

#           ActionGroup:    # ボタンをグループ化する
#               text: '種類'
#               ActionButton:
#                   text: '関数電卓'
#               ActionButton:
#                   text: 'プログラマー'


    TextInput:  # 数字表示部分
        id: display_input
        size_hint_y: 1  #  縦の大きさを全体の  1/5.5  の割合で表示する
        font_size: 60
        hint_text: '0'

    Keyboard:   # テーンキーの表示部分
        size_hint_y: 3.5    #  縦の大きさを全体の  3.5/4.5  の割合で表示する


<Keyboard@GridLayout>:  # class Keyboard(GridLayout):と同じ意味

    # 4 列×5 行のボタンを作成する
    cols: 4
    rows: 5

    spacing: 2
    padding: 4

    # 1 列目
    ClearButton:    # ボタンの種類
        text: "C"   # ボタンの表示名
    CalcButton:
        text: "%"
    DelButton:
        text: "<x"
    OperatorButton:
        text: "/"
```

```
# 2 列目
NumberButton:
    text: "7"
NumberButton:
    text: "8"
NumberButton:
    text: "9"
OperatorButton:
    text: "*"

# 3 列目
NumberButton:
    text: "4"
NumberButton:
    text: "5"
NumberButton:
    text: "6"
OperatorButton:
    text: "-"

# 4 列目
NumberButton:
    text: "1"
NumberButton:
    text: "2"
NumberButton:
    text: "3"
OperatorButton:
    text: "+"

# 5 列目
CalcButton:
    text: "+/-"
NumberButton:
    text: "0"
CalcButton:
    text: "."
```

```
    EqualButton:
        text: "="

<ButtonFormat@Button>:
    font_size: '30dp'
    background_normal: ''
    background_down: ''
    background_color: get_color_from_hex("#83481F")
    on_press: self.background_color = get_color_from_hex("#825534")
    on_release: self.background_color = get_color_from_hex("#823600")



<NumberButton@ButtonFormat>:
    #font: "Roboto"
    font_size: '30dp'
    bold: True
    on_press:  app.root.ids['calculator1'].print_number(self.text)   # ボタンが押さ
れたとき main.py の  Calculator クラスの print_number()が実行される

<OperatorButton@ButtonFormat>:
    on_press:  app.root.ids['calculator1'].print_operator(self.text)

<ClearButton@ButtonFormat>:
    on_press:  app.root.ids['calculator1'].clear_display()

<DelButton@ButtonFormat>:
    on_press:  app.root.ids['calculator1'].del_char()

<EqualButton@ButtonFormat>:
    on_press:  app.root.ids['calculator1'].calculate()

<CalcButton@ButtonFormat>:  #小数点を表示する予定だが未実装
    on_press:  app.root.ids['calculator1'].print_point(self.text)



<Calculator2@BoxLayout>
    id: calculator2
```

```
    orientation: "vertical" # オブジェクトを水平に配置
    display: display_input

    ActionBar:
        ActionView:
            ActionPrevious:
                title: '電卓 2'
                with_previous: False # true だと頭のロゴがボタンになる？
            ActionOverflow:

            ActionButton:
                text: '切り替え'
                on_press: print("push2 ");app.root.change_calc()


    TextInput:  # 数字表示部分
        id: display_input
        size_hint_y: 1  #  縦の大きさを全体の  1/4.5  の割合で表示する
        font_size: 60
    Keyboard2:   # テーンキーの表示部分
        size_hint_y: 3.5    #  縦の大きさを全体の  3.5/4.5  の割合で表示する


<Keyboard2@GridLayout>:  # class Keyboard(GridLayout):と同じ意味

    # 4 列×5 行のボタンを作成する
    cols: 4
    rows: 5

    spacing: 2
    padding: 4

    # 1 列目
    ClearButton2:    # ボタンの種類
        text: "削除"   # ボタンの表示名

    CalcButton2:
        text: "剰余"
```

```
DelButton2:
    text: "<x"
OperatorButton2:
    text: "商"

# 2 列目
NumberButton2:
    text: "七"
    calc_val: '7'
NumberButton2:
    text: "八"
    calc_val: '8'
NumberButton2:
    text: "九"
    calc_val: '9'

OperatorButton2:
    text: "積"

# 3 列目
NumberButton2:
    text: "四"
NumberButton2:
    text: "五"
NumberButton2:
    text: "六"
OperatorButton2:
    text: "差"

# 4 列目
NumberButton2:
    text: "一"
NumberButton2:
    text: "二"
NumberButton2:
    text: "三"
OperatorButton2:
    text: "和"
```

```
    # 5列目
    CalcButton2:
        text: "+/-"
    NumberButton2:
        text: "零"
    CalcButton2:
        text: "."
    EqualButton2:
        text: "="

<ButtonFormat2@Button>:
    font_size: '30dp'
    background_normal: ''
    background_down: ''
    background_color: get_color_from_hex("#398133")
    on_press: self.background_color = get_color_from_hex("#73FF66")
    on_release: self.background_color = get_color_from_hex("#52824E")


<NumberButton2@ButtonFormat2>:
    bold: True

<OperatorButton2@ButtonFormat2>:

<ClearButton2@ButtonFormat2>:

<DelButton2@ButtonFormat2>:

<EqualButton2@ButtonFormat2>:

<CalcButton2@ButtonFormat2>:
```

解説

Kv Language の基本的な説明は前の章で行ったので、ここでは説明しきれなかった部分を中心に説明します。
まずは Kv ファイルから説明します。

起動時のレイアウトについて

起動時は、rootWidget の CalculatorRoot には calculator 設定されています。
その中で calculator1 は以下のコードになっています。

```
<calculator1>
# 画面全体のレイアウト
    orientation: "vertical" # オブジェクトを水平に配置
    display: display_input

    ActionBar:

        ActionView:
            ActionPrevious:
                title: '電卓'
                with_previous: False # true だと頭のロゴがボタンになる？
            ActionOverflow:

            ActionButton:
                text: '切り替え'
                on_press: print("push ");app.root.change_calc2()
    TextInput:  # 数字表示部分
        id: display_input
        size_hint_y: 1  #  縦の大きさを全体の  1/4.5  の割合で表示する
        font_size: 60
        hint_text: '0'

    Keyboard:   # テーンキーの表示部分
        size_hint_y: 3.5    #  縦の大きさを全体の  3.5/4.5  の割合で表示する
```

calculator1 がメインのレイアウトでこのうち、TextInput が表示領域を Keyboard がキーボードの部分でこれがメインのレイアウトになります。
TextInput の「size_hint_y」の値が「1」で Keyboard の「size_hint_y」の値が「3.5」となっており、TextInput とで Keyboard の縦方向の表示領域はこの比率に沿って配置されています。

TextInput について

TextInput は文字入力ができる wighet です。今回は `hint_text: '0'` というプロパティを使用しています。`hint_text:` は未入力時の文字の表示を設定できます。
参考

- TextInput(API Reference)（https://kivy.org/docs/api-kivy.uix.textinput.html）

そのうち Kewbord は GridLayout を元にして作成したカスタム widget です。

```
<Keyboard@GridLayout>:
```

GridLayout について

公式サイトから転載

GirdLayout はグリッド（格子）上に widget を配置することが出来る widget です。

```
<Keyboard@GridLayout>:  # class Keyboard(GridLayout):と同じ意味


    # 4 列×5 行のボタンを作成する
    cols: 4
    rows: 5
```

基本的な使用方法としては「cols」が横方向を、「rows」が縦方向のセルの数を設定します。両方設定する必要はなく、「cols」のみ設定することも可能です。
今回はキーボードを作成することに使用しました。
また今回は使用していませんがセルごとに「size_hint」を使用できるので、そうすることでグリッドのサイズを個別に調整することが可能です。
なお、BoxLayout を組み合わせて使用しても同様のレイアウトは実現できますので、
GirdLayout を使用するか BoxLayout を使用するかは個人の好みによると思います。

参考

- Grid Layout(API Reference)（https://kivy.org/docs/api-kivy.uix.gridlayout.html）

ActionBar について

ActoonBar は android のアクションバー（Action Bar）と同じように、タイトルやアイコンを表示したり、ボタンを追加したりできます。おもに画面の上部、下部に設定します。


```
    ActionBar:

        ActionView:
            ActionPrevious:
                title: '電卓'
                with_previous: False # true だと戻るボタン
            ActionOverflow:

            ActionButton:
                text: '切り替え'
                on_press: print("push ");app.root.change_calc2()

            ActionButton:
                text: '音'
                icon: 'atlas://data/images/defaulttheme/audio-volume-high'

            ActionGroup:    # ボタンをグループ化する
                text: '種類'
                ActionButton:
                    text: '関数電卓'
                ActionButton:
                    text: 'プログラマー'
```

ActionView は BoxLayout を base にしたクラスで、使用してボタンなどを並べます。
ActionPrevious は、アプリのタイトルと戻るボタンを設定します。「title」プロパティに名前を、「with_previous」のプロパティを「True」にすると、kivy のアイコンをクリックすると前のページに戻ることが出きます。ただし、戻る機能を使用するには slide などで複数の画面を作成する必要があります。
またアイコン画像は「app_icon」パラメータを指定することで変更可能です。
「ActionButton」を使用するとボタンを追加します。「icon」パラメータで画像を変更できます。※icon の画像指定に「atlas」を使用しています。「atlas」は次の項目で説明します。

「ActionGroup」を使用するとボタンをグループ化します。実行結果は以下の通りです。

参考

- ActionView(API Reference)（https://kivy.org/docs/api-kivy.uix.actionbar.html#kivy.uix.actionbar.ActionView）

atlas について

atlas(アトラス)とは複数の画像を 1 つの画像にまとめたもののことです。
ゲームや CG などでよく使用する技術で、メモリの節約や読み込みを短くするために使用します。詳しい説明は「Unity　アトラス」などで検索すると出てきます

kivy でも atlas を使用しており、デフォルトのボタンや on/off スイッチなどはすべてこの仕組みを使用しております。
Kivy の atlas は atlas 形式(拡張子　.atlas)のファイルとその内部で指定された png ファイルで実現します。atlas ファイルは json 形式で以下のようになります.

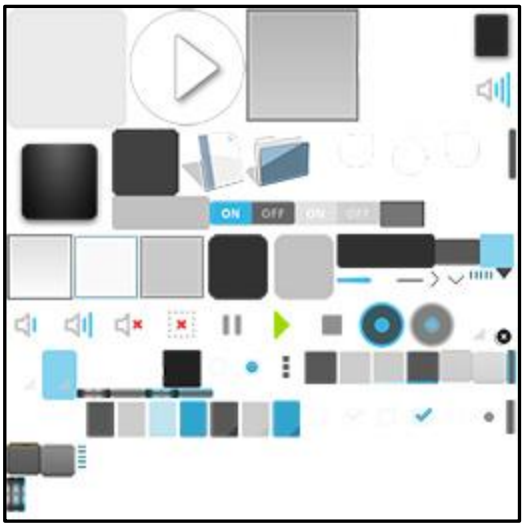
```
{
    "<basename>-<index>.png": {
        "id1": [ <x 座標>, <y 座標>, <幅>, <高さ> ],
        "id2": [ <x 座標>, <y 座標>, <幅>, <高さ> ],
        # ...
    },
    # ...
}
```

実際のファイルは以下の通りです。

kivy/data/images/defaulttheme.atlas

```
{
    "defaulttheme-0.png": {
        "action_bar": [
            158,
            133,
            36,
            36
        ],
        "action_group": [
            452,
            171,
            33,
            48
        ],
        "action_group_disabled": [
            2,
            121,
            33,
            48
        ],
        ～省略～
        "audio-volume-high": [
            464,
            406,
            48,
            48
        ],
     ～省略～
    }
}
```

defaulttheme-0.png



atlas ですが、個人的な印象だとデスクトップアプリを作成する場合だと、自分で用意して使用することはあまりないと思います。理由としてはメモリや読み込みが早くなる利点はありますが、反面ファイルの管理が面倒になるのと、現在の PC の標準的なスペックから見ても、atlas を使用せずに個々の画像ファイルから読み込んで使用しても実動作には十分だと思います。
RPG のマップ用に小さな画像を画面全体でタイル状に読み込むやシューティングゲームの弾幕のように大量に画像を表示するなどない限りは使用しなくてもよいかと思います。

参考

- Atlas(API Reference)（https://kivy.org/docs/api-kivy.atlas.html）

Kv ファイルでの複数行実行について

```
ActionButton:
    text: '切り替え'
    on_press: print("push ");app.root.change_calc2()
```

on_press 時のコマンドを見ると print コマンドの後、「;」があります。「;」を書くことで、複数のコマンドを実行できます。またこれは「;」のあと、改行＋スペースで行をそろえることもできます。

```
on_press:
    app.ans = eval(input.text);
    input.text = str(app.ans)
```

get_color_from_hex()による色の取得について

```
    #: import get_color_from_hex kivy.utils.get_color_from_hex

    background_color: get_color_from_hex("#83481F")
    on_press: self.background_color = get_color_from_hex("#825534")
    on_release: self.background_color = get_color_from_hex("#823600")
```

get_color_from_hex()を用いることで色を 16 進表記から取得できます。
get_color_from_hex()kivy.utils から import しますが、Python 側では import していません。

```
#: import get_color_from_hex kivy.utils.get_color_from_hex
```

と書くことで、Kv ファイル側で import できます。

参考

- Utils(API Reference)（https://kivy.org/docs/api-kivy.utils.html）

解説 2

次に Python ファイル側の解説をします。
Python ファイル側は前回の基本的な説明の内容でほとんど説明が可能です。一つだけ新規の要素があります。それは widget の切り替えです。

widget の切り替え

以下該当するコードの抜粋は以下の通りです。

```
from kivy.factory import Factory

class Calculator1(BoxLayout):
    clear_bool = BooleanProperty(False)

    def print_number(self, number):
       ～  省略～

#class Calculator2(BoxLayout):
#    def __init__(self, **kwargs):
#        super(Calculator2, self).__init__(**kwargs)
```

```
class CalculatorRoot(BoxLayout):

  def change_calc(self):
        self.clear_widgets()
        self.add_widget(Calculator1())

    def change_calc2(self):
        self.clear_widgets()
        calc2 = Factory.Calculator2()
        #calc2 = Calculator2()

        self.add_widget(calc2)
```

また kv ファイルは以下の通りです

```
<Calculator2@BoxLayout>
   ～省略～
```

change_calc()での関数ですが、 clear_widgets()ですが、これは該当する子の widget をすべて削除します。特定の widget のみを指定して削除する場合は remove_widget('widget 名')を使用します。反対に add_widget('widget 名')を指定することで widget を追加できます。
change_calc()は一旦、CalculatorRoot に属している widget をすべて削除（画面をクリア）して、新たに Calculator1 クラスの widget を追加します。

次に change_calc2()ですが、同様に画面をクリアして新たに Calculator2 の widget を追加していますが、change_calc()とは大きな違いがあります。
それは **Calculator1 クラスは Python ファイル内で定義していますが、Calculator2 クラスは Python 側で定義していません。kv ファイル側で定義しています**。

```
self.add_widget(Calculator2())
```

Calculator2()を Calculator1()と同じように直接実行すると起動時にエラーになります。

```
from kivy.factory import Factory

calc2 = Factory.Calculator2()
```

そこで今回使用するのは Factory クラスです。 Factory クラスを使用すると**任意のクラスまたはモジュールを自動的に登録し、プロジェクト内の任意の場所にあるクラスをインスタンス化することが可能です。そのため kv ファイルで定義した、widget を使用できます。**

この方法を使用すると、画面だけを使用したい場合で機能実装がいらない場合、Python ファイルに widget を記載しなくても済むのでプログラムが簡潔に書けます。
今回の場合ですと Factory クラスを使用しない場合、コメントアウトしていたように、Python ファイル内で Calculator2()クラスを宣言して、初期化などを宣言する必要があります。

```
class Calculator2(BoxLayout):
    def __init__(self, **kwargs):
        super(Calculator2, self).__init__(**kwargs)
```

参考

- Programming Guide(翻訳済み) » Widgets(翻訳済み)（https://kivy.org/docs/api-kivy.uix.widget.html）
- Factory object (API Reference)（https://kivy.org/docs/api-kivy.factory.html）

なお、画面を切り替える方法ですが、後は、Carousel や ScreenManager を使用する方法があります。Carousel に関しては実際にやられている方もいるので参考に挙げておきます。ScreenManager の使用方法は次回説明します。

参考

- 【修正】スライドっぽいものを作る - Kivy Advent Calendar 2013（http://d.hatena.ne.jp/cheeseshop/20131205/1386221441）
- Carousel (API Reference)（https://kivy.org/docs/api-kivy.uix.carousel.html）
- ScrollView(API Reference)（https://kivy.org/docs/api-kivy.uix.scrollview.html）

まとめ

この章で簡単な一枚の画面でおわるアプリは作成できるようになるかとおもいます。
次の章では Kivy で WebAPI の使用の仕方、ScreenManager を用いた画面遷移について説明します。

3章　～WebAPIとの連携（リクエストの送受信から結果表示まで）～

あらまし

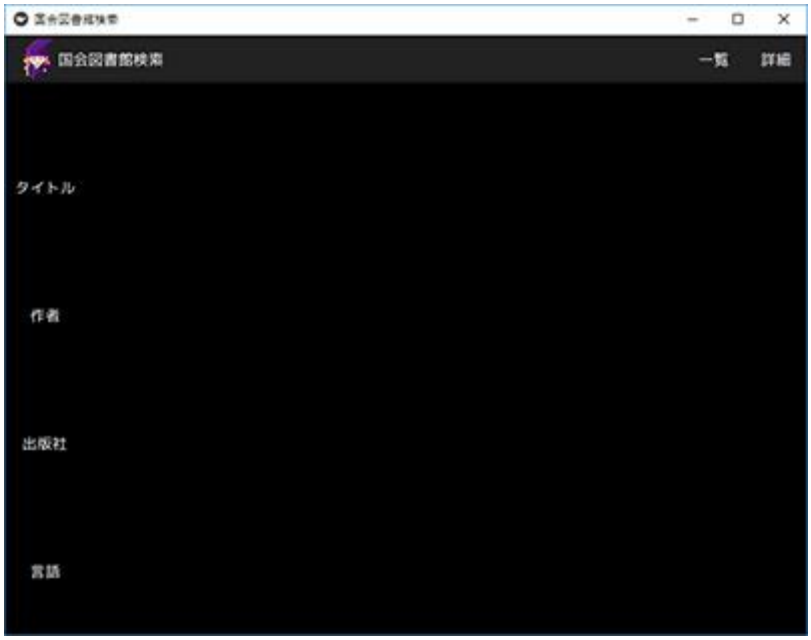
「Kv Language の基本」、「電卓を作成する」を読むと Kivy で簡単な画面が 1 枚のアプリの作成ができるようになったかと思います。
今回はマルチ画面（2 画面）のアプリを作ります。

作成するもの

今回、作成するものは WebAPI を使用して検索条件とその結果を送受信して、結果を一覧表示、一覧から選択した項目の詳細な内容を表示するようにします。
以下、実際の画面です。

起動画面（一覧表示）

詳細画面

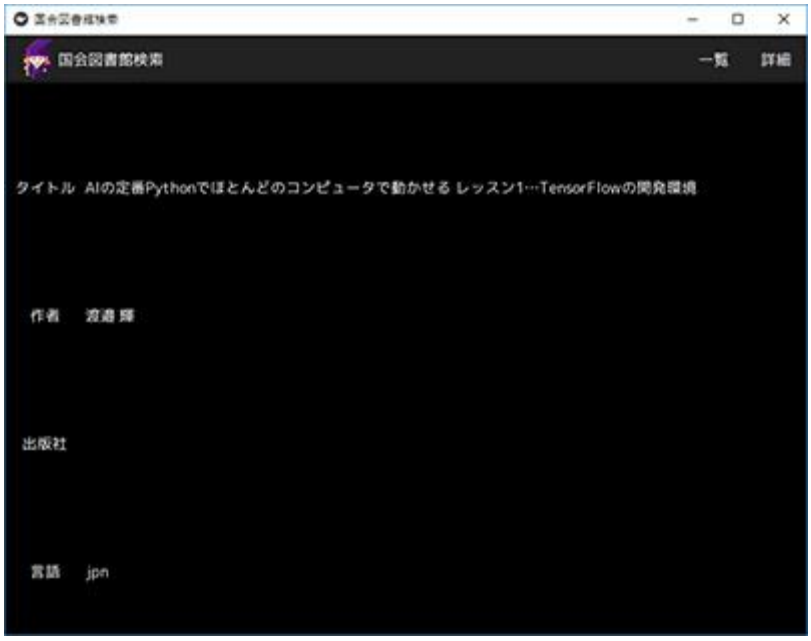


一覧表示から、右に画面をドラッグまたはアクションバーの「詳細」ボタンをクリックすると、詳細画面が横からスライドして表示されます。
検索していないので項目は何も表示されません。

実際の検索結果

検索フォームに条件を入力して、「検索」ボタンを押した結果です。
検索フォームの下に検索結果の一覧が書籍名の一覧表示で表示されます。
項目をクリックすると、自動で詳細画面に遷移します。

詳細画面（検索結果選択時）



詳細画面に書籍名とそれに付随する情報が表示されます。
画面を左ドラッグ、またはアクションバー右上の「一覧」ボタンを選択すると、 一覧画面に戻ります

一覧画面（検索語、詳細画面から遷移時）

一覧画面に戻ると、一覧から選択した項目が選択状態（赤枠）になっているのがわかります。

あらたに習得する内容

あらたに習得する内容は大きく、以下の通りです

- Carousel によるマルチ画面の作り方
- Spinner によるリスト選択の作り方
- Llistview による一覧表示の作り方※注)
- ListButton による一覧表示からのボタンの作りかた
- Adapters による、検索結果から List へのコンバートの仕方

※注) **ListView（https://kivy.org/docs/api-kivy.uix.listview.html）は次期バージョンの 1.9.2 から廃止され、代わりに　RecycleView（https://kivy.org/docs/api-kivy.uix.recycleview.html）を使用することが推奨されています。**

ただし今回は 1.9.2 がリリースされていないことと、RecycleView の仕様が発揮ししていないために、ListView を使用します。
同様に Adapters も開発中などの理由で、現在は推奨されていませんが、代変えの API が提示されていないのでこちらを使用します。

参考リンク

参考にしたものをあげます。

- Creating Apps in Kivy - O'Reilly Media
（http://shop.oreilly.com/product/0636920032595.do）

オライリーが 2014 年に出版した Kivy の入門書です。Kivy の基本的な使い方から、OpenWeatherMap（天気情報を API で提供しているサービス）を使用して、リクエストを飛ばし、各都市の天気予報を取得し表示するのを使用して結果を表示する。さらに Android、IOS などのモバイルアプリに書き出すところまでやっています。
この本が邦訳されていれば国内で Kivy の使い方がよくわからないという声は上がらなかったのではと思います。

- Kivy Tutorial 4 : Kivy ListView & ListAdapte
（https://www.youtube.com/watch?v=dxXsZKmD3Kk&t=997s）

ビデオによる ListView についての解説です。

検索する内容について

今回は国会図書館のサーチの外部提供 API（http://iss.ndl.go.jp/information/api/）を使用します。CQL という検索クエリを組み立てて送信すると蔵書結果が返ってきます。ただし、一から CQL を組み立てるのは結構手間なのと、今回は結果の送受信したデータの取扱いをメインとしたいのでこちらを使用します。
ライブラリの中身は Python で CQL の検索クエリを組み立てて、Python のライブラリである requests を使用して検索結果を送受信して結果表示しています。

参考

- Python で国立国会図書館サーチ API を使用
（http://qiita.com/noco/items/c2d1f4308ee7a50c462e）
- pyndlsearch(ソース)（https://github.com/nocotan/pyndlsearch）

検証環境

検証環境は以下の通りです。

OS: Windows10 64bit
Kivy: 1.9.1

Python3.4※ ※Kv Language ファイルですが、”Shift-jis”で保存してください。

以下、実際にコードと結果を載せていきます。

コード

Python 側のコード(main.py)は以下の通りです。

```
#-*- coding: utf-8 -*-

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.properties import ObjectProperty, ListProperty
from kivy.uix.listview import ListItemButton
from kivy.core.text import LabelBase, DEFAULT_FONT
from kivy.resources import resource_add_path



# 国立国会図書館サーチ外部提供インタフェース（API）
from pyndlsearch.client import SRUClient
from pyndlsearch.cql import CQL

# デフォルトに使用するフォントを変更する
resource_add_path('./fonts')
#resource_add_path('/storage/emulated/0/kivy/calc/fonts')
LabelBase.register(DEFAULT_FONT, 'mplus-2c-regular.ttf') #日本語が使用できるように
日本語フォントを指定する



class BookButton(ListItemButton):
    ''' search_results（ListView）の項目をボタンにする '''
    book = ListProperty()

class SearchBookForm(BoxLayout):
    search_input = ObjectProperty()
```

```
    search_results = ObjectProperty()   # kv ファイル側の search_results（ListView）
を監視

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


    def books_args_converter(index, data_item):
        ''' 検索結果を book 名をキーとした辞書型に変換する。
             検索結果のレコード分呼ばれて実行される。
        '''

        title, creater , language, publisher = data_item
        return {'book': (title, creater , language, publisher )}

    def search_book(self):
        ''' 検索条件をもとに検索し、結果を ListView に格納する '''

        print('search_book')

        cql = CQL()

        # ★検索条件を入力していく
        cql.title = self.search_input.text

        year  = self.ids['year'].text
        month = self.ids['month'].text
        day = self.ids['day'].text


        cql.fromdate = year + '-' + month + '-' + day # 出版年月日
        #cql.fromdate = '2000-10-10'
        #print(cql.payload())
        #cql.title = 'Python'
        #cql.fromdate = '2000-10-10'
        # NDL Search クライアントの設定

        client = SRUClient(cql)
```

```
        client.set_maximum_records(int(self.ids['number'].text))  #最大取得件数
        # ★検索条件入力終了

        #client.set_maximum_records(10)  #最大取得件数
        #print(client)

        # get_response()では xml 形式で取得可能
        #res = client.get_response()
        #print(res.text)

        # SRU 実行（入力条件を元に検索を実行する）
        srres = client.get_srresponse()

        # 検索結果を books リストに格納
        books = [(d.recordData.title, d.recordData.creator, d.recordData.language,
d.recordData.publisher) for d in srres.records]
        print(books)

        print("----------------")
        # 検索結果に格納する

        self.search_results.adapter.data.clear()       # 検索結果を data（詳細表示
用）を消去
        self.search_results.adapter.data.extend(books) # 検索結果を data に追加
        self.search_results._trigger_reset_populate()  # search_results
（list_view）をリフレッシュ


class BookInfo(BoxLayout):
    ''' 詳細画面の情報 '''
    book = ListProperty(['', '','',''])

class BookSearchRoot(BoxLayout):

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

    def show_book_info(self, book):
```

```
        ''' 選択した情報を整形して詳細画面へ移動して表示する '''
        print('BookSearchRoot')

        print(book) #Book = BookButton()の値、取れているか確認用

        # Label の Text に None が入るとエラーになるために変換を行う
        book_convs = [x if x != None else '' for x in book] # None が返ってきた場合は""に変更する

        # 詳細画面に書籍情報を格納
        self.bookinfo.book = book_convs

        # 詳細画面に移動
        self.carousel.load_slide(self.bookinfo)

class BookSearchApp(App):

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

        self.title = '国会図書館検索'
    pass

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

Kv ファイル

Kv Language（BookSearch.kv）は以下の通りです。

```
#: import main main
#: import ListAdapter kivy.adapters.listadapter.ListAdapter

# 起動時に表示される widget
BookSearchRoot

<BookSearchRoot>
    # 一覧画面
```

```
    carousel: carousel
    booklists: booklists
    bookinfo:  bookinfo

    BoxLayout:
        orientation: "vertical"
        ActionBar:
            ActionView:
                ActionPrevious:
                    title: "国会図書館検索"
                    with_previous: False
                    app_icon: "./icon/32player.png"

                ActionButton:
                    text: "一覧"

                    # 一覧画面に移動する
                    on_press: app.root.carousel.load_slide(app.root.booklists)
                ActionButton:
                    text: "詳細"

                    # 詳細画面に移動する
                    on_press: app.root.carousel.load_slide(app.root.bookinfo)

        Carousel:
            id: carousel
            SearchBookForm:  # 一覧画面
                id: booklists
            BookInfo:          # 詳細画面
                id: bookinfo

<SearchBookForm>
    # 一覧画面のレイアウト
    orientation: "vertical"
    search_input: search_box    # ①クラス変数を追加　こうすることで Python 側で
self.search_input が取れる
    search_results: search_results_list
```

```
    # 検索フォーム
    BoxLayout:
        height: "40dp"
        size_hint_y: None
        TextInput:
            id: search_box  # ②  「①」で値を渡せる
            size_hint_x: 70
        Button:
            text: "検索"
            size_hint_x: 30
            on_press: root.search_book()

    BoxLayout:
        size_hint:1,.1

        Label:
            size_hint: .2,1
            text: "出版年月"

        Spinner:    # 年のリスト表示
            id: year
            size_hint: .1,1
            halign: 'center'
            valign: 'middle'
            text_size: self.size
            text:'2000'
            values: [str(y) for y in range(2000, 2018) ]

        Label:
            size_hint: .05,1
            text: "年"

        Spinner:
            id: month   # 月のリスト表示
            size_hint: .05,1
            halign: 'center'
            valign: 'middle'
            text_size: self.size
```

```
            text:'01'
            values: ['{0:02d}'.format(x)  for x in range(1,13)]

        Label:
            size_hint: .05,1
            text: "月"

        Spinner:    # 日のリスト表示
            id: day
            size_hint: .05,1
            halign: 'center'
            valign: 'middle'
            text_size: self.size
            text:'01'
            values: ['{0:02d}'.format(x)  for x in range(1,30)] # 月ごとに日が変わる処理が必要だが一旦保留

        Label:
            size_hint: .05,1
            text: "日"



        Label:
            size_hint: .05,1
            text: "件数"

        Spinner:    # 月ごとのサイズ
            id: number
            size_hint: .05,1
            halign: 'center'
            valign: 'middle'
            text_size: self.size
            text:'10'
            values: ['1','5','10', '15', '20']



    ListView:
        id: search_results_list
```

```
        adapter:
            # 検索結果を一覧表示かつ項目をボタンにする
            # data = 検索の一覧をリストで保持する
            # CLS  = 一覧の表示形式（今回はボタンにして表示する）
            # args_converter = 表示結果を書籍名をキーにしたリストに変換する.
            ListAdapter(data=[], cls=main.BookButton,
args_converter=main.SearchBookForm.books_args_converter)

<BookButton>
    # 検索結果をボタンにするレイアウト
    text_size: self.size
    halign: 'left'

    text: self.book[0]  #書籍名をボタンのタイトルにする

    height: "40dp"
    size_hint_y: None
    on_press: app.root.show_book_info(self.book)



<BookInfo>
    # 検索結果
    book: ("","","","")
    orientation: "vertical"


    BoxLayout:
        orientation: "horizontal"
        size_hint_y: None
        height: "40dp"


    GridLayout:
        cols: 2
        rows: 4

        Label:
```

```
            text: "タイトル"
            halign: 'left'
            valign: 'middle'
            size_hint_x: 20
            text_size:self.size

        Label:
            text_size:self.size
            halign: 'left'
            valign: 'middle'
            text:root.book[0]
            size_hint_x: 80

        Label:
            text: "作者"
            halign: 'left'
            valign: 'middle'
            size_hint_x: 20
            text_size:self.size

        Label:
            text: root.book[1]
            size_hint_x: 80
            text_size:self.size
            halign: 'left'
            valign: 'middle'

        Label:
            text: "出版社"
            halign: 'left'
            valign: 'middle'
            size_hint_x: 20
            text_size:self.size

        Label:
            text: root.book[3]
            size_hint_x: 80
            #text: "出版社:{} ".format(root.book[3])
```

```
            text_size:self.size
            halign: 'left'
            valign: 'middle'


        Label:
            text: "言語"
            halign: 'left'
            valign: 'middle'
            size_hint_x: 20
            text_size:self.size

        Label:
            text: root.book[2]
            size_hint_x: 80
            text_size: self.size
            halign: 'left'
            valign: 'middle'
```

解説

解説は、Kv Language から始めます。

「BookSearchRoot」widget について

起動時は「BookSearchRoot」widget が表示されます。「BookSearchRoot」大きく分けて 2 つの widget で成り立っています。

該当の Kv は以下の通りです。

該当のコードは以下の通りです。

```
<BookSearchRoot>
    # 一覧画面
    carousel: carousel
    booklists: booklists
    bookinfo:  bookinfo

    BoxLayout:
```

```
        orientation: "vertical"
        ActionBar:
            ActionView:
                ActionPrevious:
                    title: "国会図書館検索"
                    with_previous: False
                    app_icon: "./icon/32player.png"

                ActionButton:
                    text: "一覧"

                    # 一覧画面に移動する
                    on_press: app.root.carousel.load_slide(app.root.booklists)
                ActionButton:
                    text: "詳細"

                    # 詳細画面に移動する
                    on_press: app.root.carousel.load_slide(app.root.bookinfo)

        Carousel:
            id: carousel
            SearchBookForm:  # 一覧画面
                id: booklists
            BookInfo:           # 詳細画面
                id: bookinfo
```

Carousel について

Carousel はスワイプすることで、画面（スライド）を切り替える widget です。
今回は、2 画面（SearchBookForm,BookInfo）を操作します。使い方は widget 名と id を使用して操作します。また以下のようにすることで見せたい画面を自動的にスワイプします。

```
carousel.load_slide(表示したいスライドの id)
```

他にも切り替わりの速度の変更などのプロパティがあるので詳しくは API リファレンスを参考にしてください。

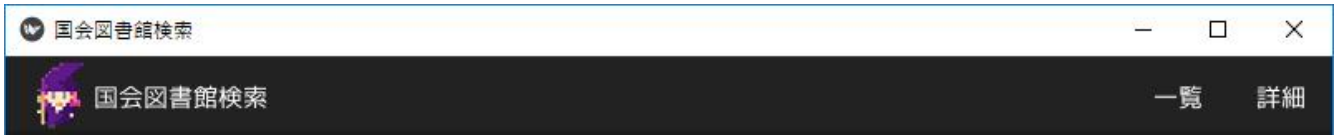
参考

- Carousel(API Reference)（https://kivy.org/docs/api-kivy.uix.carousel.html）

前回は画面の切り替わりに「clear_widgets()」を使用して画面(widget)を一旦削除し、「add_widgets()」を使用して画面(widget)を作成しました。
このやり方だと問題としては**前の画面に戻るのが難しいです。**
widget を削除してしまうので値の保持も難しいです。Carousel を使用することで各画面の値を保持したまま、複数の画面に遷移して行き来できます。

アクションバー

画面上部のアクションバーは以下の画面になっています。



該当のコードは以下の通りです。

```
        ActionBar:
            ActionView:
                use_separator: True
                ActionPrevious:
                    title: "国会図書館検索"
                    with_previous: False
                    app_icon: "./icon/32player.png"

                ActionButton:
                    text: "一覧"

                    # 一覧画面に移動する
                    on_press: app.root.carousel.load_slide(app.root.booklists)
                ActionButton:
                    text: "詳細"

                    # 詳細画面に移動する
                    on_press: app.root.carousel.load_slide(app.root.bookinfo)
```

このうち画面左側のタイトル部分ですが、今回は App_icon に独自のアイコンを指定して表示させています。アイコン素材は元の Kivy のアイコンのサイズが 32px×32px なのでサイズはそれに習いました。

またボタンを押すことで、各画面に移動できます。

SearchBookForm について

SearchBookForm は大きく分けて、2 つに分かれます。 1 つは、検索条件を入力する検索フォーム部分、もう一つは検索結果を一覧表示する部分です・

検索フォーム部分について

画面は以下の通りです。

またフォームを構成している Kv 部分は以下の通りです。

```
# 検索フォーム
BoxLayout:
    height: "40dp"
    size_hint_y: None
    TextInput:
        id: search_box  # ②  「①」で値を渡せる
        size_hint_x: 70
    Button:
        text: "検索"
        size_hint_x: 30
        on_press: root.search_book()

BoxLayout:
    size_hint:1,.1

    Label:
        size_hint: .2,1
        text: "出版年月"

    Spinner:    # 年のリスト表示
        id: year
        size_hint: .1,1
        halign: 'center'
        valign: 'middle'
```

```
            text_size: self.size
            text:'2000'
            values: [str(y) for y in range(2000, 2018) ]

        Label:
            size_hint: .05,1
            text: "年"

        Spinner:
            id: month   # 月のリスト表示
            size_hint: .05,1
            halign: 'center'
            valign: 'middle'
            text_size: self.size
            text:'01'
            values: ['{0:02d}'.format(x)  for x in range(1,13)]

        Label:
            size_hint: .05,1
            text: "月"

        Spinner:    # 日のリスト表示
            id: day
            size_hint: .05,1
            halign: 'center'
            valign: 'middle'
            text_size: self.size
            text:'01'
            values: ['{0:02d}'.format(x)  for x in range(1,30)] # 月ごとに日が変わる
処理が必要だが一旦保留

        Label:
            size_hint: .05,1
            text: "日"



        Label:
            size_hint: .05,1
```

```
            text: "件数"

        Spinner:    # 月ごとのサイズ
            id: number
            size_hint: .05,1
            halign: 'center'
            valign: 'middle'
            text_size: self.size
            text:'10'
            values: ['1','5','10', '15', '20']
```

このうち、TextInput()については、文字入力部分で前章・前々章で説明しました。
ちなみに WindowsOS の文字入力で IME が開かないのは Kivy のバグです。 日本語で検索する場合はメモ帳などで入力した値をコピペして使用してください。

次に、年月日の一覧を選択するのに今回は「Spinner」を使用しています。Spinner はクリックするとリストが表示され、そこから項目を選択することができます。

年の該当の Kv は以下の通りです。

```
Spinner:    # 年のリスト表示
    id: year
    size_hint: .1,1
```

```
    halign: 'center'
    valign: 'middle'
    text_size: self.size
    text:'2000'
    values: [str(y) for y in range(2000, 2018) ]
```

Values に入れた値がリストになります。

詳しくは API リファレンスを参考にしてください。

Spinner

- Carousel(API Reference)（https://kivy.org/docs/api-kivy.uix.spinner.html）

一覧画面表示部分について

次に、検索結果の一覧表示をしている部分の説明をします。

。

表示を構成している Kv 部分は以下の通りです。

```
    ListView:
        id: search_results_list
        adapter:
            # 検索結果を一覧表示かつ項目をボタンにする
            # data = 検索の一覧をリストで保持する
            # CLS  = 一覧の表示形式（今回はボタンにして表示する）
            # args_converter = 表示結果を書籍名をキーにしたリストに変換する.
            ListAdapter(data=[], cls=main.BookButton,
args_converter=main.SearchBookForm.books_args_converter)
```

ここではまず ListView について説明します。

ListView

ListView はデータをリスト形式で表示するための widget です。
簡単な使い方は、「item_strings」と呼ばれるプロパティにリスト構造を入力します。
例えば以下のコードの場合、item_strings には 0 から 100 までの連続した数が入り、実行すると、0 から 100 までの項目が Label 形式で表示されます。

```
class MainView(ListView):
    def __init__(self, **kwargs):
        super(MainView, self).__init__(
            item_strings=[str(index) for index in range(100)])
```

ただし、この形式で表示されるのはあくまでも Label なので選択して動作をすることができません。
そこで今回は、Adapters（https://kivy.org/docs/api-kivy.adapters.html）というメソッドを使用して一覧をボタン表示に変えます。

```
ListAdapter(data=[], cls=main.BookButton,
args_converter=main.SearchBookForm.books_args_converter)
```

引数の説明は以下の通りです。

- data：検索結果の情報をリストで保持する
- CLS： 一覧 index にして表示します（今回はボタンに設定して表示）
- args_converter = 検索結果のレコードを 1 件ずつ書籍名をキーにしたリストに変換する

各項目の Python 側のファイルと照らし合わせて説明します。

data

data は検索結果の値を保持します 今回ですと検索を行うと以下の用なタプルが返ってきます。

該当のコードは以下の通りです。

```
    def search_book(self):
        ''' 検索条件をもとに検索し、結果を ListView に格納する '''

    ～ 省略 ～

        # SRU 実行（入力条件を元に検索を実行する）
        srres = client.get_srresponse()
```

```
        # 検索結果を books リストに格納
        books = [(d.recordData.title, d.recordData.creator, d.recordData.language,
d.recordData.publisher) for d in srres.records]
        print(books)

        print("----------------")
        # 検索結果に格納する

        self.search_results.adapter.data.clear()       # 検索結果を data（詳細表示
用）を消去
        self.search_results.adapter.data.extend(books) # 検索結果を data に追加
        self.search_results._trigger_reset_populate()  # search_results
（list_view）をリフレッシュ
```

books リストに格納される値は以下の通りです。

```
[('10 日でおぼえる Python 入門教室', '穂苅実紀夫，寺田学，中西直樹，堀田直孝，永井孝 著
', 'jpn', '翔泳社'),

～省略～

 ('Bluetooth 接続のおもちゃを制御する ロボット専用ツールで操作 Python で自動化', None,
'jpn', '')]
```

以下の該当の箇所で data の前回の結果を削除して、新たに配置しています。
その後「_trigger_reset_populate」で表示を更新しています。

```
    self.search_results.adapter.data.clear()       # 検索結果を data（詳細表示用）を
消去
    self.search_results.adapter.data.extend(books) # 検索結果を data に追加
    self.search_results._trigger_reset_populate()  # search_results（list_view）を
リフレッシュ
```

args_converter

```
def books_args_converter(index, data_item):
    ''' 検索結果を book 名をキーとした辞書型に変換する。
         検索結果のレコード分呼ばれて実行される。
    ''
    title, creater , language, publisher = data_item
    return {'book': (title, creater , language, publisher )}
```

args_converter は検索結果の書籍情報をキーにして args_converter に格納しています。

cls

```
class BookButton(ListItemButton):
    ''' search_results（ListView）の項目をボタンにする '''
    book = ListProperty()
```

cls は一覧の表示形式を設定します。今回は書籍の情報を List 化して book に格納して Button に表示します。

表示の形式に関しては Kv 側の以下のプロパティを使用します。

```
<BookButton>
    # 検索結果をボタンにするレイアウト
    text_size: self.size
    halign: 'left'

    text: self.book[0]  #書籍名をボタンのタイトルにする

    height: "40dp"
    size_hint_y: None
    on_press: app.root.show_book_info(self.book)
```

Adpter の概念、使用は難しくて私にも完全には把握できていませんが、「data」,「cls」,「args_converter」に値を設定することで一覧の表示形式の設定を変更できることがわかれば一旦大丈夫です。

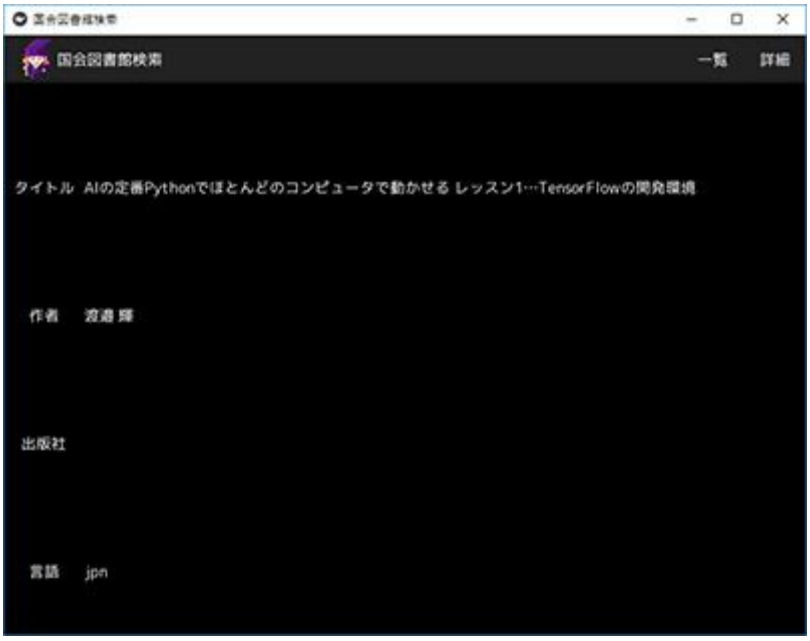
参考

- List View(sample code)（https://github.com/kivy/kivy/blob/master/examples/widgets/lists/list_simple.py）
- List View(API Reference)（https://kivy.org/docs/api-kivy.uix.listview.html）
- Adapters(API Reference)（https://kivy.org/docs/api-kivy.adapters.html）

「BookInfo」widget について

詳細画面の説明です。

画面は以下の通りです。



Kv ファイルは以下の通りです。

```
<BookInfo>
    # 検索結果
    book: ("","","","")
    orientation: "vertical"


    BoxLayout:
        orientation: "horizontal"
        size_hint_y: None
        height: "40dp"


    GridLayout:
        cols: 2
        rows: 4

        Label:
```

```
            text: "タイトル"
            halign: 'left'
            valign: 'middle'
            size_hint_x: 20
            text_size:self.size

        Label:
            text_size:self.size
            halign: 'left'
            valign: 'middle'
            text:root.book[0]
            size_hint_x: 80

        Label:
            text: "作者"
            halign: 'left'
            valign: 'middle'
            size_hint_x: 20
            text_size:self.size

        Label:
            text: root.book[1]
            size_hint_x: 80
            text_size:self.size
            halign: 'left'
            valign: 'middle'

        Label:
            text: "出版社"
            halign: 'left'
            valign: 'middle'
            size_hint_x: 20
            text_size:self.size

        Label:
            text: root.book[3]
            size_hint_x: 80
            #text: "出版社:{} ".format(root.book[3])
```

```
            text_size:self.size
            halign: 'left'
            valign: 'middle'


        Label:
            text: "言語"
            halign: 'left'
            valign: 'middle'
            size_hint_x: 20
            text_size:self.size

        Label:
            text: root.book[2]
            size_hint_x: 80
            text_size: self.size
            halign: 'left'
            valign: 'middle'
```

「BookButton」widget をクリックして main.show_book_info()が実行されます。
show_book_info()では book の情報を整形して、「BookInfo」widget（詳細画面）が表示されます。

ここでは結果を表示するだけで特に難しいことをやっていません。
唯一新規でやっているのは文字を左端にそろえて折り返して表示することでしょうか。

```
Label:
    text_size:self.size
    halign: 'left'
    valign: 'middle'
    text:root.book[0]
    size_hint_x: 80
```

Label のテキストは、デフォルトのままだと、長い行だと Label からはみ出して 1 行に表示されるため、「text_size」をラベルと同じサイズに設定して折り返しをありにしています。そのうえで「halign」で横方向の表示を左寄せ、「 valign」で縦方向の表示を真ん中に表示します。

参考

- Kivy の Label の text_size 属性
  （http://qiita.com/gotta_dive_into_python/items/7715cc838819329c2a20）
- Kivy Label の左寄せ、右寄せについて
  （http://qiita.com/Agipy/items/17b9631313f40d80687b）

まとめ

今回で、複数の画面の表示、複数の入力項目を元にした検索、結果の一覧表示、各項目の詳細表示の方法がわかったかと思います。今回までボタンを使用したデスクトップアプリに関しては作り方がなんとなくわかったかと思います。

次の章ではこのプログラムを元に Android 端末で表示してみます。

ちなみに、ここから Android に表示するまでにプログラムを大きく変更する必要があり、結構大変でした。

参考　ファイルのパッケージ化について

Kivy を使用してファイルのパッケージ化をしたい場合は PyInstaller をしようしてパッケージ化ができます。windows で exe 化をしたい場合は別途 **PyWin32** をインストールする必要があります。

参考

- Kivy と PyIntstller のパッケージ化について（http://www.slideshare.net/JunOkazaki1/kivypyintstller）
- Windows10 で、Kivy を使う（https://torina.top/main/299/）
- Programming Guide(翻訳済み) » Create a package for Windows（翻訳済み）（https://pyky.github.io/kivy-doc-ja/guide/packaging-windows.html）

4章　～Android での実行～

あらまし

この章では、Kivy が Qt 系のライブラリと比較した際に利点とされる、モバイル（iOS,Android） 系の出力のうち Android の出力について説明します。先に結論から言うと個人的には微妙です。帯に短しタスキに長しというか、こちらがたつと後ろが立たずというか・・・。

作成するもの

前の章の『WebAPI との連携（リクエストの送受信から結果表示まで）』を Android 端末で実行してみます。

使用するコード

前回のコードと内容はほぼ一緒です。変更があった部分に関しては、適宜紹介します。

Kivy を Andorid 端末で実行する方法

方法としては 3 つ方法がありますが目的によって実行方法が違います。

①自分の保有する Android 端末で動かす
②APK を作成し、Google Play などで配布して不特定多数の人に利用してもらう
③python-for-android を利用して android 端末上で Python を実行する

①の場合は、Kivy Launcher（https://play.google.com/store/apps/details?id=org.kivy.pygame&hl=ja）という、Android アプリを使用して実現します。今回はこの方法を採用して説明します。

②の場合は Ubuntu に Buildozer というパッケージをインストールして実現します。Windows の場合は、VM が配布されていますのでそちらを使用してください。https://kivy.org/#download の Virtual Machine の項目からダウンロードできます。

実際に使用する場合は、Kivy アプリケーションを Goole Play Store で公開する為に #2 APK を作成（http://qiita.com/gotta_dive_into_python/items/f3a6a2c83ed96366bc2f）という記事で実際に実行された方がいるので参考にしてみてください。

③の場合は①、②とは方法性が違って Android 端末上で実際に Python スクリプトを実行するという方法になります。Qpython で検索すると色々出てきます。

この辺りの内容ですが、2016 年現在の「Python を android で使う」（http://hhsprings.pinoko.jp/site-hhs/2016/08/2016%E5%B9%B4%E7%8F%BE%E5%9C%A8%E3%81%AE%E3%80%8Cpython-%E3%82%92-android-%E3%81%A7%E4%BD%BF%E3%81%86%E3%80%8D/）という記事で詳しく説明されていますので、興味のある方は読んでください。

参考

- Programming Guide(翻訳済み) » Create a package for Android(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/packaging-android.html）
- QPython - Python on Android（http://qpython.com/）

実行する Python 環境について

検証環境ですが、Kivy は 1.9.1 になります。一方 Python の方は **Python2.7** になります。Kivy の公式サイトを見ると、Android は Python3 に暫定的に対応したとありますが、2017 年 2 月現在は実質的に Python2 系対応で Python3 系には非対応と考えたほうが良いです。理由は後述します。
なお iOS は Python2.7 対応で Python3 系の対応は現在対応中だそうです。

実際に検証に使用した Android 端末について

試した機種は以下の 2 種類になります。

- Nexus (OS Android6.01)
- ZenFone 3 Laser(OS Android6.01)

Android 自体はメーカー、機種ごとに固有の機能があるので機種、OS によっては動かないかもしれないです。

Android で Kivy を実際に動かす

いか、実際の操作になります。

Kivy Launcher について

Kivy Launcher（https://play.google.com/store/apps/details?id=org.kivy.pygame&hl=ja）は Google Play で配布されているアプリで、使用すると Android 上で Kivy を使用した Python スクリプトを手軽に使用できます。 以下簡単な使用方法です。

なお、使用方法を説明するにあたり、Kivy 公式サイトで配布している demo プログラムである、「touchtracer」を用いて説明します。

参考

- Gallery of Examples(翻訳済み) » Touch Tracer Line Drawing Demonstration(翻訳済み)（https://pyky.github.io/kivy-doc-ja/examples/gen__demo__touchtracer__main__py.html）
- ソースコード(github)（https://github.com/kivy/kivy/tree/master/examples/demo/touchtracer）

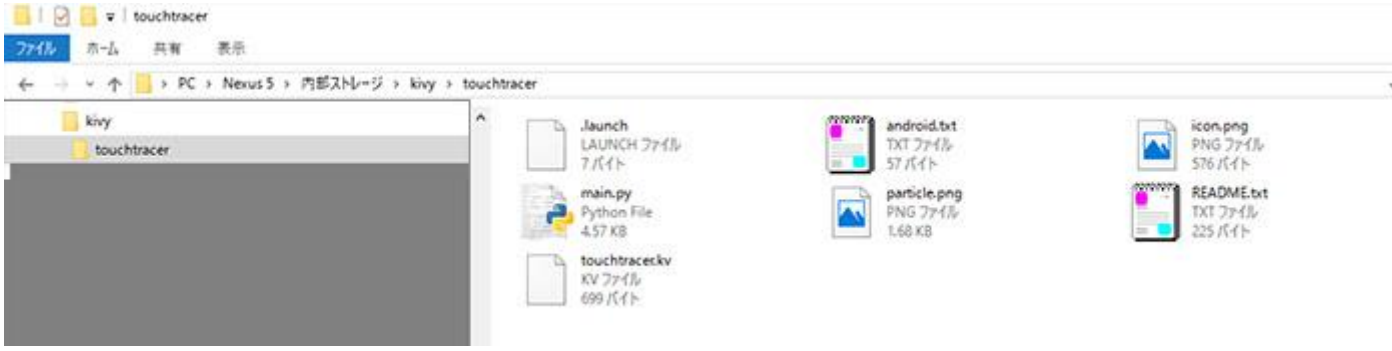
Kivy Lancher の使い方

事前に、Android に Play ストアから KivyLancher をインストールしてください

①Andorid 端末を PC と USB 接続します。

ファイル転送を選択してください。

②Kivy フォルダを作成してファイルを配置する



「Kivy」というディレクトリを作成します。
ディレクトリ配下にフォルダを作成しプログラムを置いてください。

置くのは以下のファイルです。

- main.py :実行する Python ファイルです。**ファイル名は必ず main.py にしてください**
- kv ファイル
- android.txt：アプリの情報を各ファイルです。**必須ファイルです。**
- icon.pic :アプリのメニュー一覧で表示される画像ファイルです。必須ではないです。

android.txt ですが中身は以下の通りです

```
title=Touchtracer
author=Kivy team
orientation=landscape
```

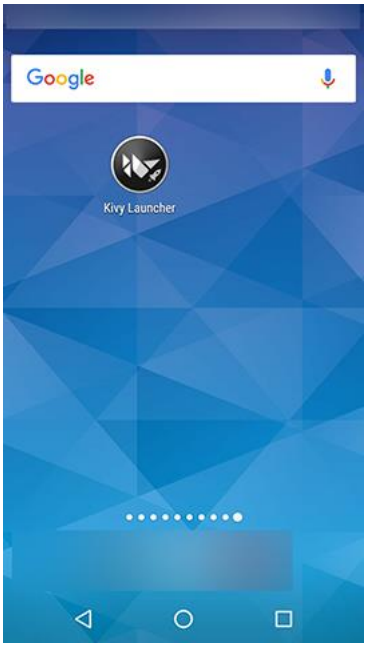
「title」にはアプリの名前を「author」には作者名をいれてください。
「orientation」は起動時に縦、横どちらの向きを表示するかです。「landscape」では横向きを、「portrait」では縦の向きになります。

③アプリを起動して動かす

実際に動かしてみます。

1. Kivy Lancher を実行する

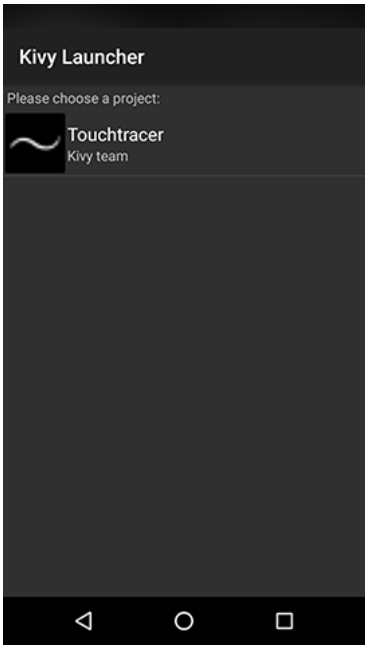
Android 端末で Kivy Lancher を実行します。



待ち画面が表示されます。

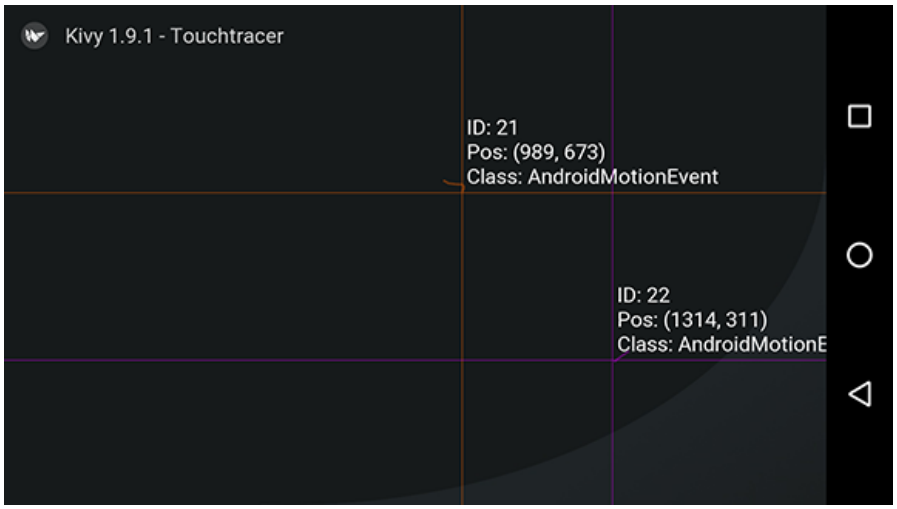
2. アプリを選択する

アプリの一覧が表示されるので、アプリを選択してください。



3. アプリの起動画面

選択後、待ち画面が表示され、しばらく立つと選択したアプリが端末上で動作できます。



前回のアプリを動してみる

前回のアプリを実際に動かしてみます。「Touch tracer」のように動かそうとしますが、Kivy Lancher 上から選択して起動しようとしてもうまく動かず、アプリ自体がクラッシュしてしまいます。 Android 本体の.kivy の内部の Log を回収してみました。以下がログの中身の抜粋です。

```
[INFO              ] Kivy: v1.9.1
[INFO              ] Python: v2.7.2 (default, Mar 20 2016, 23:30:13)
```

Python2.7 で実行されていました・・・。 そのために、前回作成していたプログラムを動かすためには、Pytho3 から Python2 系に変換しないといけませんでした。具体的には以下の変換を行いました。 なお筆者は Python に関しては 3 系から始めたので 2 系はほとんどやっていませんでした。

- 検索結果の data(list 型)の初期化を del list[:]に変更する。
- 日本語の文字列周りのエンコードエラーの対応のために「.encode('utf-8')」を適宜入れる。
- Requests ライブラリが使用できなかったため Kivy 独自のライブラリ UrlRequest を使用し、それに関する修正を行う。

このうち、最後の UrlRequest の修正について説明します。

Kivy の UrlRequest の使い方

ソースで修正した箇所を以下に抜粋します。

```
from kivy.network.urlrequest import UrlRequest
import urllib

class AddSearchBookForm(BoxLayout):
    def search_book(self):
        set_title = urllib.quote((self.search_input.text).encode('utf-8'))
        req_url =
"http://iss.ndl.go.jp/api/sru?operation=searchRetrieve&query=title%3d%22{0}%22%20
AND%20from=%22{1}%22&maximumRecords={2}&mediatype=1".format(set_title,
set_fromdate, set_maximum_records)
```

```
        res = UrlRequest(req_url, self.gotBooklists)

    def getBooklists(self, req, results):
        result_records = result_parse(results)
```

UrlRequest ですが、今回の修正に伴う特徴は 2 つあります。

- URL エンコードが必要。
- UrlRequest は**非同期通信**である。

UrlRequest（https://kivy.org/docs/api-kivy.network.urlrequest.html）の API リファレンスを見ると以下のような仕様になっています。

```
req = UrlRequest(url, on_success, on_redirect, on_failure, on_error,
                 on_progress, req_body, req_headers, chunk_size,
                 timeout, method, decode, debug, file_path, ca_file,
                 verify)
```

まず URL は%文字列である必要があるため、urllib で%文字列に変換して第一引数にいれました。
また実行時に返却する値の req には url からの POST した返却結果が格納されますですが非同期通信のために、サーバから結果が返らず、すぐには値が入ってこないのでプログラムが先に進んでしまいます。そのため前回のままだと、POST した結果の処理が上手くいかないという問題があります。
第 2 引数の on_success に、サーバーサイドからの POST 結果が返ってきた場合の処理（メソッド）を設定するか、または返り値に wait()を使用して、返却結果が戻るのを待つ必要があります。今回は前者の on_success を使用する方法を用いて、getBooklists()を on_sucess に設定し、getBooklists()に結果をパースして整形をおこなう処理を実行するようにしました。

参考

- In kivy python how to get data from a url request(stackoverflow)（http://stackoverflow.com/questions/38152228/in-kivy-python-how-to-get-data-from-a-url-request）
- UrlRequest(API Reference)（https://kivy.org/docs/api-kivy.network.urlrequest.html）

実行結果

Pytho2 系の対応をして、動作を行うと Kivy Lancher 上でプログラムが実行できました。

参考　Android 特有の API を実行する方法

今回は使用しませんが、Kivy では Android 固有の JAVA の API に関しては Plyer と Pyjnius を使用することで実行できます。これを使用することで、アプリ内で例えばカメラを起動して写真を撮影するやメールを送るといったことが可能です。Android アプリで標準的な機能は Plyer または Pyjnius を使用することで実現できます。

- **Plyer** Python ラッパーのプラットフォーム非依存 API 群。
- **Pyjnius** Python から Java / Android API への動的アクセスする。

実際に国内で使用されている方の記事と公式サイトのリンクをあげておきます

- pyjnius を使って写真を撮る - Kivy Advent Calendar 2013
  （http://d.hatena.ne.jp/cheeseshop/20131214/1386987210）

- plyer.compass が使い物にならない件（http://hhsprings.pinoko.jp/site-hhs/2016/07/plyer-compass-%E3%81%8C%E4%BD%BF%E3%81%84%E7%89%A9%E3%81%AB%E3%81%AA%E3%82%89%E3%81%AA%E3%81%84%E4%BB%B6/#more-8424）
- Using Android APIs(Android API を使用)（https://pyky.github.io/kivy-doc-ja/guide/android.html#using-android-apis）
- Play (github)（https://github.com/kivy/plyer）
- Pyjnius(github)（https://github.com/kivy/pyjnius）

参考　iOS へ対応について

iOS に関しては筆者は試したことがないですが、iOS アプリの開発も可能です。
コードは Python2 系対応のみとなり Pytho3 系は未対応です。
toolchain という機能を使用して Xcode 用のプロジェクトを作成し、そのプロジェクトからビルドできるようになるようです。

参考

- Python のマルチプラットフォーム GUI ライブラリ kivy を使って Mac, Ubuntu, iOS 用の GUI アプリを作る方法（http://myenigma.hatenablog.com/entry/2016/05/06/170854）
- Programming Guide(翻訳済み) » Creating packages for OS X(翻訳済み)（https://pyky.github.io/kivy-doc-ja/guide/packaging-osx.html）

iOS 特有の API に関してはこれもまた Pyobjus を使用することで、アプリ内で使用できるそうです。

参考

- Pyobjus（https://github.com/kivy/pyobjus）

国内では小樽商科大学の原口先生が Kivy アプリを作成して AppStore に出されて言います。
原口先生はまた Kivy の公式マニュアルの翻訳(https://pyky.github.io/kivy-doc-ja/)の主要メンバーの一人です。

参考 小樽商科大学　原口研究室（http://puzzle.haraguchi-s.otaru-uc.ac.jp/）

参考　Python の外部ライブラリを使用するには

モバイルで Matplotlib などの Python の標準ライブラリ以外を使用するには Kivy garden というアドオンの機能を使用して実現することになります。

参考

- kivyLauncher　環境でグラフ　(+ kivy garden　について)（http://hhsprings.pinoko.jp/site-hhs/2016/08/kivylauncher-%E7%92%B0%E5%A2%83%E3%81%A7%E3%82%B0%E3%83%A9%E3%83%95-kivy-garden-%E3%81%AB%E3%81%A4%E3%81%84%E3%81%A6/）
- Kivy Garden (github)（https://github.com/kivy-garden/）

参考　Kivy で作成されたアプリについて

公式サイトの Wiki を見ると Kivy を使用して作成された、アプリプロジェクトの一覧があります。
これを見ると実際にどのようなアプリが作成可能かがわかるかと思います。

参考

- List of Kivy Projects（https://github.com/kivy/kivy/wiki/List-of-Kivy-Projects）

まとめ

サーバーサイドと連携して表示するアプリを Android 端末で動かすことが実際にできました。Android のアプリに関しては 2D 画像主体のアプリですと、カード系のソーシャルゲームや検索アプリなど、大体の物は開発が可能だと思います。

個人的には Android アプリを作成するにあたり、Kivy を使用するかどうかは微妙なところです。Android アプリを出すということは iOS 版の開発も視野にいれてやるのが主流かと思います。そうなると Pytho2 系での開発が前提になりますが、日本語（マルチバイト文字）の扱いに手間がかかります。日本でのコミュニティもないので、Xamarin や Unity といった C#を使ったマルチプラットフォームの開発や、Andori ネィテイブで開発する方が現時点で

はよいのかなと思っています。 少なくとも国内での Python エンジニアの数と相まって、積極的に採用するものではないと思います。

Python3 系の対応が iOS でも正式にリリースされ、Kivym Lancher のような簡易のプレビューアプリができればまた話は変わってくるとは思いますが、現段階ではこのような評価になります。

全体のまとめ

全 4 章にわたって Kivy の使い方について説明しました。まだ説明していない主要な機能としては Animation がありますが機会があれば触れようと思っています。
今までの説明を読めば一般的なアプリを開発する事が可能、もしくは取っ掛かりにはなると思います。また Kivy がどんなライブラリで何ができて何ができないかもおぼろげながらに分かってきたかと思います。

Kivy の利点と欠点を上げておくと以下になるかと思います。

利点

- Python でアプリが書ける
- Kv Language を使えばレイアウトの変更が簡単にできる

欠点

- TextInput のバグで OS によっては日本語入力で IME が開かない
- マルチプラットフォームと銘打っているが iOS/Android 対応になると Python2 系での開発が必要
- 同様に、標準ライブラリ以外を使用する場合には工夫が必要

このあたりを踏まえて Kivy を使用するかしないかを考えてみてもらえると幸いです。
またバグに関してはオープンソースなので issue や要望を上げていく、改良してコミットするということで改善はされていくのでそこはユーザー次第かと思います。

参考

- User's Guide(翻訳済み) » Contributing(翻訳済み)（https://pyky.github.io/kivy-doc-ja/contribute.html）
- User's Guide(翻訳済み) » Contact Us(翻訳済み)（https://pyky.github.io/kivy-doc-ja/contact.html）

Kivy に関しては現在も github 上で活発なやり取りをされていてオープンソースとしての将来性はあると思っています。
本書を読んで Kivy を使用してみたいと思いましたら幸いです。

奥付

作成：2017 年 2 月 14 日　初版作成

著者：オカザキ （twitterID:@dario_okazaki）

編集：創 （twitterID:@sow_an）

問い合わせ：http://studio-overdrive.com/?page_id=209

制作：StudioOverdrive(http://studio-overdrive.com/)