Kivyを用いたマルチタッチアプリ開発

原口　和也

Kivy フレームワークは，Python でマルチタッチアプリを作ることのできる希少なツールの一つである．これを用いてプログラムを作れば，Linux，macOS，Windows などの主要なデスクトップ OS はおろか，タブレットやスマートフォンで用いられる Android や iOS で動くアプリを生成することができる．本稿ではこの Kivy フレームワークの基本的な使い方を説明する．

キーワード：Kivy フレームワーク，マルチタッチアプリ，クロスプラットフォーム

1. はじめに

Kivy は Python でマルチタッチアプリを開発するためのフレームワークである．最初のバージョン 1.0.0 は 2011 年 2 月にリリースされた．本稿執筆時点での最新バージョンは 2018 年 7 月にリリースされた 1.10.1 である．

Kivy の魅力は Python でアプリを作ることができること，MIT ライセンスの下で（ほかの権利関係に抵触しない限り）商用配布可能なこと，そして**クロスプラットフォーム**であろう．すなわち主要 OS の多くで開発環境を構築することができ，またアプリ化することができるのである．公式サイト [1] によると，Kivy のインストールがサポートされている OS は Windows (7, 8, 10), macOS (10.9 以降), Linux (Ubuntu, Mageia, Arch, Fedora, OpenSUSE), Android (2.2 以降)，Raspberry Pi，Slackware である．また Windows, macOS, Ubuntu では Anaconda 環境を用いることも可能である．さらに Windows, macOS, Linux, Android，iOS においてアプリ化することができる．

プログラミング面では，**関心の分離** (separation of concerns) の思想が取り入れられていることが大きな特徴である．**KV 言語**という独自言語を用いることにより，機能面を Python，GUI 面を KV 言語でおおよそ分けて書くことができる．原理的には Python だけですべてを書くこともできるが，そのプログラムは冗長で，管理しづらいものとなるだろう．KV 言語は決して難しいものではなく，スタイルシート (CSS) のような形式によって GUI のデザインを設定することができる．

本稿の目標は，簡単なプログラムの作成を通じて，Kivy プログラムの開発イメージを読者に掴んでもらうことである．Python の使用は必要最低限に留め，ほぼすべての記述を KV 言語で行う．KV 言語だけでどの程度の GUI を実現できるかを感じてもらいたい．本稿で取り上げるプログラムは，macOS Sierra 10.12.6, Python 2.7.10, Kivy 1.10.0 で動作を確認している．

Kivy は PyPI (Python Package Index) に登録されているので，pip を用いてインストールすることができる[1]．ただし使用する環境の構成によってはインストールがうまくいかないかもしれない．その場合は公式ドキュメンテーションを参照したり，ウェブで最新の情報を入手するなどして解決することを勧める．

2. GUI の構成

2.1 ウィジェットツリー

Kivy の GUI は，ウィジェット（部品）を組み合わせることによって構成される．ライブラリには多数のウィジェットクラスが提供されていて，以下の 3 種類に大別される．

パーツ：最小単位のウィジェット．文字列を表示するための `Label` や，タッチ操作を受け付けるための `Button` などが提供されている．

レイアウト：ほかのウィジェットを規則的に配置するためのウィジェット．ウィジェットを一列に並べるための `BoxLayout` や，座標を自由に決めて配置するための `FloatLayout` などが提供されている．

スクリーンマネージャ：複数の画面を保持し，表示を切り替えるためのウィジェット．紙芝居風の `Carousel` や，Android 風の `ActionBar` などが提供されている．

はらぐち　かずや
小樽商科大学商学部社会情報学科
〒 047–8501 北海道小樽市緑 3–5–21
haraguchi@res.otaru-uc.ac.jp

[1] Ubuntu の場合，PPA を用いてインストールすることもできる．

所望のウィジェットを生成し，それらの間に適切な親子関係を定めることで，GUI が構成される．たとえばいくつかのパーツを横一列（もしくは縦一列）に配置するには，これらのパーツをすべて `BoxLayout` オブジェクトの子にする．また `BoxLayout` オブジェクトから成る画面をいくつか保持し，それぞれを紙芝居風に切り替えたいのであれば，これら `BoxLayout` オブジェクトを，`Carousel` オブジェクトの子とすればよい．

ウィジェット間の親子関係から定まる木構造を**ウィジェットツリー**といい，根に相当するウィジェットを**ルートウィジェット**という．一般に葉に対応するウィジェットはパーツで，内点に対応するウィジェットはレイアウトやスクリーンマネージャとなる．

2.2 “Hello, world.”

一般的な Kivy プログラムは，(1) Python スクリプト（*.py ファイル），(2) KV スクリプト（*.kv ファイル），(3) 画像や音声などのアセットからなる．Python スクリプトを実行すればプログラムが起動する．

“Hello, world.” をウィンドウ上に表示するためのプログラムを示す．まず Python スクリプトを示す．

```
from kivy.app import App
class MyApp(App):
    pass
MyApp().run()
```

インポートしている `App` は基本クラスである．一般的な Kivy プログラムではこの `App` のサブクラスを定義して用いる．本稿ではこのサブクラスをアプリクラスと呼ぶ．上の例では `MyApp` がこれにあたる．アプリクラスの名前には `...App` のように末尾に `App` を付けるのが慣例である．アプリクラスのオブジェクトを生成して `run()` メソッドを実行すると，**メインループ**が開始する．メインループではイベントの検知やグラフィックスのレンダリングが繰り返し行われる．メインループが終了すると，プログラムも終了する．

上記の Python スクリプトは必要最低限の構成である．つまり，実質的な部分のすべてを KV 言語で書く場合であっても，Python スクリプトとして上の 4 行（もしくはそれに準ずる表現）は必要である．**本稿では Python スクリプトとして上記のみを用いる**こととし，以後このスクリプトを `main.py` と呼ぶ．

次に KV スクリプトを示す．KV スクリプトのファイル名は（アプリ名）`.kv` としておくと，プログラム起動時に自動的に読み込んでくれるので便利である[2]．（アプリ名）は，アプリクラス名の末尾の `App` を（もしあれば）取り除き，残った文字列をすべて小文字にしたものである．上記の `main.py` の場合，アプリ名は `my`，自動的に読み込まれる KV ファイルの名前は `my.kv` となる．

```
Label:
    text: 'Hello, world.'
    font_size: 24
    color: 0,1,0,1
```

上記の KV スクリプトでは，ルートウィジェットである `Label` オブジェクトに関する**ウィジェットルール**が定められている．一般に KV スクリプトは，ルートウィジェットもしくは個々のクラスについて，主にデザインに関する初期値をルールとして定めるものである．ルートウィジェットに関するルールをウィジェットルール，個々のクラスに関するルールを**クラスルール**という．一つの KV スクリプトにおいて，ウィジェットルールは高々一つしか書くことができないが，クラスルールはいくつ書いてもよい．クラスルールの例は後述する．

再び上記の KV スクリプトに戻る．このプログラムのウィジェットツリーは `Label` オブジェクト一つからなる．その `Label` オブジェクトの文字列 (`text`) は `'Hello, world.'`，フォントサイズ (`font_size`) は 24，文字の色 (`color`) は緑である（RGBA 形式によって指定）．Python 同様，KV 言語では入れ子構造をインデントによって表す．何個の半角空白文字によってインデント 1 単位とみなすかは，KV スクリプトの最初に現れるインデントにしたがって決定される．

2.3 レイアウトの使用例

次にレイアウトの使用例を示す．ここで取り上げるレイアウトは，縦もしくは横にウィジェットを並べる `BoxLayout` である．以下に KV スクリプトを示す．また図 1 にスクリーンショットを，図 2 にウィジェットツリーを示す．このように KV 言語では，ウィジェット間の親子関係を入れ子構造によって表す．

```
1  BoxLayout:
2      orientation: 'vertical'
3      Label:
4          text: 'Hello, world.'
5          bold: True
6      Button:
7          text: 'Press me.'
8      TextInput:
9  <Label>:
10     font_size: 24
```

[2] 本稿における KV スクリプトは，すべてこの流儀にしたがってファイル名が付けられたものとする．

図 1　BoxLayout のスクリーンショット

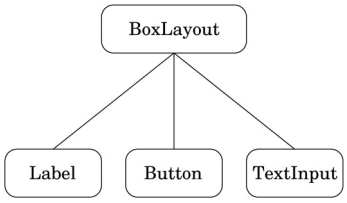

図 2　BoxLayout のウィジェットツリー

KV スクリプトの 2 行目，

```
    orientation: 'vertical'
```

は，子ウィジェットを縦に並べるための設定である（横ならば'horizontal'）．また 9, 10 行目は，Label クラスに対するクラスルールである．クラスルールの書き出しは

```
<クラス名>:
```

である．クラスルールは，当該クラス（この場合 Label）およびそのサブクラスのオブジェクトすべてに適用される．詳細は省略するが，ライブラリの中では Button は Label のサブクラスとして設計されているため，フォントサイズを 24 にするというクラスルールは，Label オブジェクトと Button オブジェクトの双方において適用される．一方太字 (bold) は，Label オブジェクトについてのみ設定されている（5 行目）．

2.4　スクリーンマネージャの使用例

次にスクリーンマネージャの使用例を示す．ここでは紙芝居風に画面を切り替えるための Carousel を用いる．以下に KV スクリプトを示す．また図 3 にスクリーンショットを，図 4 にウィジェットツリーを示す．

```
1  Carousel:
2      BoxLayout:
3          orientation: 'vertical'
4          Label:
5              text: 'Hello, world.'
6              bold: True
7          Button:
8              text: 'Press me.'
9          TextInput:
10             id: txtbox
11     Label:
12         text: 'You typed \"' + txtbox.text +
               '\".'
13 <Label>:
14     font_size: 24
```

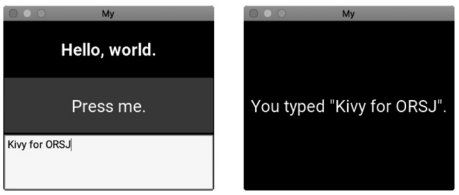

図 3　Carousel のスクリーンショット．二つの画面はドラッグ操作（もしくはスワイプ操作）によって切り替えることができる．

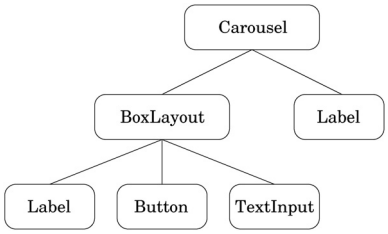

図 4　Carousel のウィジェットツリー

Carousel オブジェクトの子である BoxLayout オブジェクトと Label オブジェクトが，それぞれ一つの画面を構成する．二つの画面はドラッグ操作（タブレットやスマートフォンならばスワイプ操作）によって切り替えることができる．

KV スクリプトの 9, 10 行目に注目してもらいたい．

```
        TextInput:
            id: txtbox
```

ここでは TextInput オブジェクトに id を与えている（その値は txtbox）．ここで与えられた id は，同一ルール内のほかのオブジェクトから参照することができる．11, 12 行目を見てみよう．

```
    Label:
        text: 'You typed \"' + txtbox.text +
              '\".'
```

Label オブジェクトの文字列 (text) が，txtbox によって参照される TextInput オブジェクトの text プ

ロパティ，すなわち入力された文字列を用いて定められている．これにより，TextInput オブジェクトに入力された文字列が，自動的に Label オブジェクトの文字列に反映されるのである．

このように KV 言語では，あるプロパティ p の初期値を，別のプロパティ q を用いて定めておくと，q の値が変わったときに p の値も自動的に更新される．これについては次節で改めて述べる．

3. イベントの処理

Kivy プログラムは，**イベント**にしたがってその流れが決定される，イベント駆動型のプログラムとみなすことができる．Kivy が取り扱うことのできる主なイベントは，以下の三つである．

- **ウィジェットイベント**：「ボタンが押される」など，ウィジェットごとに定義されたイベント．
- **クロックイベント**：「10 秒後に 1 度だけ発生」「0.5 秒ごとに発生」のように，実時間に基づいて発生するイベント．
- **タッチイベント**：デバイスからの入力に対して発生するイベントのうち，座標情報をもつもの．たとえばスクリーンタッチやマウス操作など．

本節ではこのうちウィジェットイベントについて概説する．

3.1 ボタン

Button クラスでは，「押される」イベントに対して on_press メソッド，押されていたのが「離される」イベントに対して on_release メソッドがそれぞれ紐づけされている．イベントが発生したときの処理を記述するには，それぞれのメソッドをオーバーライドすればよい．以下の KV スクリプトでは，ボタンが押されると “You pressed me.”，押されていたのが離されると “You released me.” が端末に出力される．

```
Button:
    text: 'Press me.'
    on_press:
        print('You pressed me.')
    on_release:
        print('You released me.')
```

このように KV スクリプトには，イベント処理のための **on メソッド**を記述することができる．ただし上の例のようにインデントレベルが増えない簡単なものに限られる．また（名前が on で始まらない）一般のメソッドを記述することはできない．KV スクリプトに書けないようなメソッドは，Python スクリプトに書くことになる．

3.2 プロパティ

Kivy ではウィジェットに**プロパティ**という属性をもたせることができる．このプロパティは Kivy 独自の仕組みであり，Python における property 関数やデコレータとはまったく異なるものである．本稿でこれまで取り扱ってきた，Label クラスの text, font_size, color などもすべてプロパティである．

プロパティは**オブザーバーパターン**の機能を備えていて，プロパティに値が代入されるとき，Kivy はその値が妥当であるか否かを検証する．もし値が妥当な場合，プロパティの値が変更されたというイベント（**プロパティイベント**）が発生する．一方，もし代入しようとする値が妥当でない場合，例外が発生する．

自分で新しくプロパティを定義する場合は，そのプロパティで取り扱う値に応じて適切なプロパティクラスを選択する必要がある．プロパティクラスには一般の数値を取り扱うための NumericProperty，最大値と最小値をもつような数値を取り扱うための BoundedNumericProperty，文字列を取り扱うための StringProperty，リストを取り扱うための ListProperty などがあるが，KV スクリプトでプロパティを定義する場合，初期値に応じて適切なプロパティクラスが自動的に選択される[3]．

```
<MyWidget>:
    my_prop1: 123
    my_prop2: 'orsj'
```

上の例では，my_prop1 のプロパティクラスは数値を取り扱うための NumericProperty，my_prop2 のプロパティクラスは文字列を取り扱うための StringProperty となる．以後，my_prop1 に数値以外の値（あるいは my_prop2 に文字列以外の値）を代入しようとすると，例外が発生する．

前節で述べたとおり，KV 言語においてプロパティ p の初期値を別のプロパティ q を用いて定めると，q の値が更新されたとき，p の値も自動的に更新される．

以下は数を数えるカウンターのための KV スクリプトである．図 5 にこのプログラムのスクリーンショットを示す．

```
1 BoxLayout:
2     counter: 0
3     Label:
4         size_hint_x: 3
5         text: str(root.counter)
```

[3] Python 側で定義する場合はプロパティクラスを自分で選ぶ必要がある．

図 5　カウンタープログラムのスクリーンショット

```
    Button:
        text: 'Press me.'
        on_press:
            root.counter += 1
<Label>:
    font_size: 24
```

BoxLayout オブジェクトに対し，プロパティ counter が定められている（2 行目）．初期値が 0 であることから，この counter のプロパティクラスは NumericProperty となる．画面右側のボタンをタッチすると，counter の値は 1 増える（9 行目）．なお root は KV 言語におけるキーワードであり，同一ルールにおけるルートウィジェットを指す（この場合 BoxLayout オブジェクト）．

さて，5 行目では Label オブジェクトの text プロパティの値が，root.counter を用いて定められている（str(x) は値 x を文字列に変換）．これにより，counter の値が変わると，文字列も自動的に変わるのである．

なお 4 行目の size_hint_x は，幅の比を表すためのプロパティである．BoxLayout オブジェクトの幅は，size_hint_x の値に基づいて Label オブジェクトと Button オブジェクトに比例配分される．デフォルト値は 1 であるため，Label オブジェクトと Button オブジェクトの幅の比は 3 : 1 となる．

このようにウィジェットのサイズや位置を比に基づいて指定することができるが，ピクセル数や座標を用いて明示的に指定することもできる（詳細は省略）．

3.3　プロパティイベント

プロパティイベントが発生したとき，それに対して処理を行うための関数やメソッドを紐づけすることができる．当該プロパティ prop と同じクラスの中でメソッドを紐づけしたい場合，デフォルト名の on_prop を用いると便利である．

カウンタープログラムに関して，以下のように on_counter メソッドを定めると，counter の値が更新されるたびに on_counter が呼び出され，現在の counter の値が端末に出力される．

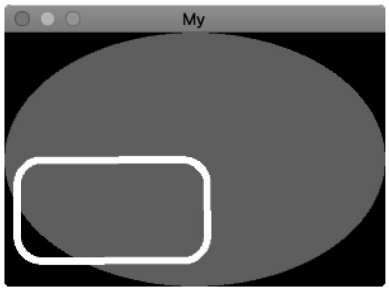

図 6　キャンバスのスクリーンショット

```
BoxLayout:
    counter: 0
    on_counter:
        print('counter =', self.counter)
    # ...
```

なお self は当該オブジェクト（この場合 BoxLayout）を指す，KV 言語のキーワードである．

4.　キャンバス

任意のウィジェットはキャンバスをもっていて，その上に図形を描画することができる．描画可能な図形には，直線，三角形，四角形，多角形，楕円，真円，ベジエ曲線などがある．

描画は命令を連ねることで行うが，命令には**コンテキスト命令**と**描画命令**の 2 種類がある．コンテキスト命令は色の設定やキャンバスの回転など，描画の状況に関する命令である．一方描画命令は，定められたコンテキストの下で描画を行うための命令である．

例として，赤い楕円と角の丸い長方形を描画するための KV スクリプトを以下に示す．図 6 はスクリーンショットである．

```
Widget:
    canvas:
        Color:
            rgba: 1,0,0,1
        Ellipse:
            pos: self.pos
            size: self.size
        Color:
            rgba: 1,1,1,1
        Line:
            width: 3
            rounded_rectangle:
                10,20,150,80,20
```

2 行目以降にあるとおり，ウィジェットの canvas 属性に対して命令を書き連ねる．なおウィジェットが持つキャンバスには canvas のほか canvas.before と canvas.after があり，前者は canvas の奥に，後者

は canvas の手前に描画される．3, 8 行目の Color は色を指定するためのコンテキスト命令である．5 行目の Ellipse は楕円を，10 行目の Line は線を描くための描画命令である．この Ellipse では，self が指す Widget オブジェクトと同じ位置 (pos) に，同じサイズ (size) の楕円を描く．この pos と size はプロパティなので，Widget オブジェクトの位置やサイズが変わると，楕円のサイズもそれにつれて変わる．一方 Line では，左下の点の座標を (10, 20) とし[4]，幅 150px，高さ 80px の，角が丸い（その半径は 20px）長方形が描かれる．また線の幅は 3px である．

5. サンプルプログラム

ここまでの内容を用いて，ボタンをタッチすると画像が切り替わるようなプログラムを作ってみよう．以下に KV スクリプトを示す．また図 7 にこのプログラムのスクリーンショットを示す．

```
1  BoxLayout:
2      orientation: 'vertical'
3      filename: ''
4      GridLayout:
5          rows: 2
6          cols: 2
7          ToggleButton:
8              text: 'バルセロナ'
9              filename: 'Barcelona.jpg'
10         ToggleButton:
11             text: 'ベルリン'
12             filename: 'Berlin.jpg'
13         ToggleButton:
14             text: '石巻'
15             filename: 'Ishinomaki.jpg'
16         ToggleButton:
17             text: '小樽'
18             filename: 'Otaru.jpg'
19     Image:
20         size_hint_y: 5
21         source: root.filename
22
23 <ToggleButton>:
24     font_name: 'VL-Gothic-Regular.ttf'
25     font_size: 24
26     group: 'image'
27     filename: ''
28     on_state:
29         if self.state=='down': app.root.
               filename = self.filename
```

このプログラムのルートウィジェットは BoxLayout オブジェクトである（1 行目）．このオブジェクトは GridLayout オブジェクトと Image オブジェクトの二つを子にもつ．前者は格子状にウィジェットを並べるためのレイアウトで，後者は画像を表示するためのウィジェットである．GridLayout における格子の行数と列数は，それぞれ rows プロパティと cols プロパティによって指定する（5, 6 行目）．Image における画像ファイルの名前は，source プロパティによって指定する（21 行目）．ただしここでは source プロパティに root.filename が渡されている．これはすなわち，ルートウィジェットの filename プロパティの値である．ルートウィジェットの filename プロパティの値が更新されると，Image オブジェクトの source プロパティの値も更新されるのである．

図 7　画像表示プログラムのスクリーンショット

さて，GridLayout オブジェクトは四つの ToggleButton オブジェクトを子にもつが，ToggleButton のクラスルールが 23 行目以降に定められている．この ToggleButton は，オン（「押された」状態）もしくはオフ（「押されていない」状態）の二つの状態になりうるボタンである．

- 24 行目の font_name プロパティはフォントファイルを指定するためのもので，日本語を表示するときには不可欠である[5]．
- 26 行目の命令により，四つの ToggleButton オブジェクトに共通して group='image' が代入される．一方，group の値が等しいオブジェクトが複数存在する場合，高々一つしかオン状態になることができない．なお ToggleButton オブジェクトがオンかオフかは，state プロパティを見ればわかる．オンの場合，state プロパティの値は 'down' で，オフの場合は'normal' となる．
- 28, 29 行目について，ToggleButton オブジェクトの状態がオンになると（すなわち state='down' になると），ルートウィジェット (app.root) の filename プロパティに，当該 ToggleButton オブジェクトの filename の値が渡される．ここで

[4] Kivy の座標系はウィンドウの左下の点を原点 (0, 0) とする．

[5] ここで用いたフォントは VL ゴシック [2] である．

app は KV 言語のキーワードで，アプリオブジェクト（実行中のアプリクラスのオブジェクト）を指し，app.root でルートウィジェットを指す．ToggleButton オブジェクトの filename の値は，具体的には 9, 12, 15, 18 行目で定められている．

6. おわりに

本稿では簡単なプログラムの作成を通じて，Kivy フレームワークの基本的な使い方を説明してきた．Python の使用は必要最低限に留め（2.2 節の main.py のみ），ほぼすべての部分を KV 言語で書いてきた．

KV 言語で書けるのは GUI の部分やごく簡単な on メソッドのみに過ぎず，より複雑なプログラムを実現するには Python スクリプトにコードを書く必要がある．そもそも，1 節で述べたとおり，わざわざ KV 言語を用いなくともすべてのコードを Python だけで書くことだってできる．しかし一般には，KV 言語を用いることでプログラム全体が簡潔になり，管理がしやすくなるであろう．

本稿で触れられなかった Kivy の基本的な機能として，クロックイベント，タッチイベント，アニメーションなどがある．これらを含め，Kivy プログラミングの基本をもう少し詳しく知りたい読者は公式のドキュメンテーション [3] や拙著 [4] などを参照されたい．

関連プロジェクトとして，Buildozer や Kivy-ios（Android 端末や iOS 端末にプログラムを転送するためのツール），KivEnt（ゲームプログラミング用ツール），Garden（ユーザが開発したウィジェットやライブラリ集）などがある．関連プロジェクトについては Kivy@github [5] を参照されたい．

Kivy は未だ開発途上にあり，洗練化を待たないといけない点も少なくない．以下にその例を挙げておく．

- Python3 への対応が不完全である．標準部分についてはだいぶ対応が進められてきたが，関連プロジェクトの一部（Kivy-ios など）は未だ Python2 にしか対応していない．
- 一部の OS では TextInput で日本語を入力することができない．
- ドキュメンテーションが読みづらい．また不完全な箇所も多数ある．
- アセットが充実していない．

最後に Kivy と筆者の出会いについて触れる．筆者は指導学生とともに組合せパズルの自動生成に関する研究を行ってきたが，コンピュータに生成させたパズルを人が遊べるようにしたいと，あるとき考えた．そこでアプリの開発を目論んだのだが，プログラミング言語や IDE の使い方を新しく覚えることなく，できれば Python のような簡単な言語で楽に開発したかった．そこで出会ったのが Kivy である．

Kivy には若干のとっつきにくさはあるものの，多少慣れてしまえば学生でもある程度のアプリを開発することができる．実例として，筆者らがこれまでに Kivy を用いて開発したアプリを筆者のウェブサイト [6] で公開している．巷では高額を要求されるアプリ開発だが，実はそこまで難しくないのである．アプリ開発に対する敷居を下げ，わが国の技術力の底上げに寄与したいと考えて書籍 [4] を執筆したが，エンジニアのみならず，大学生や高専生，さらには中高生にまで読者がいるようでひそかに喜んでいる．この本では基本的な内容を中心に書いたので，より実践的な内容を次著 [7] にまとめることを企図している．

参考文献

[1] Kivy 公式サイト, https://kivy.org/（2018 年 8 月 31 日閲覧）
[2] VL ゴシック，http://vlgothic.dicey.org（2018 年 8 月 31 日閲覧）
[3] Kivy: Getting Started, https://kivy.org/docs/gettingstarted/intro.html（2018 年 8 月 31 日閲覧）
[4] 久保幹雄（監修），原口和也，『Kivy プログラミング—Python でつくるマルチタッチアプリ—』，朝倉書店，2018.
[5] Kivy@github, https://github.com/kivy/kivy（2018 年 8 月 31 日閲覧）
[6] 筆者のウェブサイト，http://puzzle.haraguchi-s.otaru-uc.ac.jp（2018 年 8 月 31 日閲覧）
[7] 久保幹雄（監修），原口和也，『実践・Kivy を用いたマルチタッチアプリ開発（仮題）』，近代科学社，近刊予定.

