# プログラミングII(Python言語) 第5週目 講義資料

## 期末課題について

期末課題の説明を参考に実施.

## GUI概要

ターミナルのように文字だけの入力によって操作するインタフェースのことをCUI(character interface)と呼ぶ．一方で普段のPC操作のように，マウス／タッチパッドなどを利用して，操作するインタフェースをGUI(graphical interface)と呼ぶ．pythonでpyQTやkivyなどのパッケージを利用してGUIを作成することができる．ここでは，kivyについて説明する.

　ちなみにPythonの標準モジュールのGUIはtkinterが存在するが，現時点でMacで利用するとMacがクラッシュする不具合が生じているため，今回は保留.

pythonのGUIパッケージの比較

　全体的にPythonのGUIパッケージはあまり決定版はない．今後状況が変わる可能性があるが，現時点でモダンとされているkivyを今回は説明する

## kivyの参考先

国内の参考書籍の多くはtkitnter（またはゲーム開発関係はpygame）を利用しているが，上記の理由でkivyを使う．ただし，現時点で日本語の参考書籍はあまり多くない．webサイトも含めていくつか紹介をしておく.

公式のリファレンス:一部のみ日本語訳あり
webサイト: Python Kivyの使い方① 〜Kv Languageの基本〜
書籍: Kivyプログラミング ―Pythonで作るマルチタッチアプリ―

## kivyの概要

Mac，Win,Linuxなどで開発が可能であり，またPC用のデスクトップアプリだけではなく，AndoroidやiOS用のアプリケーションの開発にも利用できる.
基本的な手順としては

1. ウィンドウの大きさを設定
2. パーツ（ラベルやボタンなど）を設定
3. パーツのレイアウトを設計
   という手順で画面を構築していく．(パーツのことをウィジェットと呼ぶが，本資料ではとりあえずパーツで統一)
4. ボタンなどに対するアクション（クリックするなど）
   を必要に応じて設定していく．

このような部品とレイアウトで設定する形はJavaのJavaFXなどと近い．

またこの形式のGUIパッケージは，GUIの画面を設定するファイル（スタイルファイルなどと)と実行するファイルを分けることができる．kivyでは設定ファイルをkv言語という言語で書く．
本講義では，kv言語をpythonファイル内に埋め込む形（１つのファイル）で実施するやり方で実施する．

# kivyの利用

kivyは標準モジュールではなくサードーパーティのモジュールのためinstallしてからimportする必要がある．以下ターミナルで実行する．

```
pip install kivy
```

加えて以下のモジュールもインストールしておく．

```
pip intall japanize-kivy
```

↑はimportするだけでkivyで日本語フォントが利用できるようなるパッケージ．最近リリースされたが便利なので今回の講義では利用する

## kivyの初歩

まずはHelloと表示させる画面を設定する．

```
from kivy.core.window import Window #1
Window.fullscreen = False
Window.size = (500,300)

from kivy.app import App #2a
from kivy.uix.label import Label #2b

# 3
class HelloApp(App):
    def build(self):
        label1 = Label(text="hello") #3a
        return label1

#4
HelloApp().run()
```

1. ウィンドウサイズの設定(#1の部分)
   1から３行の部分はGUIのウィンドウの大きさを設定する．現時点でMac(Anaconda由来のpython)ではデフォルトでは，フルスクリーンでGUIが作成されるので，Falseでフルスクリーンとしない．
   さらに，ウィンドウのサイズはWindow.siz(widht,hight)で設定できる(単位はピクセル)．とりあえず500,300で設定．
2. モジュールのimport(#2)
   Kivyを利用するときにはkivyパッケージからappをimportしておく（kivyのアプリケーションを起動するために必要なもの）
   さらに，パーツとしてLabelを利用するのでLabelもimportしておく（他に必要なパーツの種類があるときには，それぞれをimportしておく）
3. クラスの設計(#3)
   クラスを作成し(#3)，そのクラスを実行する(#4)という形でGUIを作成する．
   クラスの引数はAppとする（これはkivy.appからimport)されている．クラスの中の初期関数（コンストラクタ）は，基本的に，selfを引数として，内部にパーツを設定していく
   3aの部分では，Labelというクラス(impor Labelから持ってきた)．Labelのクラスの引数としてtextなどが存在するので，ここでは"hello"と設定しておく．
   Labelの他設定などは以下の公式を参照
   そして作成したLabelクラスのlabel1というオブジェクトを戻り値とする．
4. クラスの実行(#4)
   上記で作成したクラスHelloAppを実行させる．このとき実行のメソッドとしてrun()を利用する．

以下HelloAppを改変し，２つのパーツを並べ，かつ片方をボタンとしたmのである．

```
from kivy.core.window import Window #1
Window.fullscreen = False
Window.size = (500,300)

from kivy.app import App #2a
from kivy.uix.label import Label #2b
from kivy.uix.boxlayout import BoxLayout #2c
from kivy.uix.button import Button #2d

#3
class HelloApp(App):
    def build(self):
        layout = BoxLayout(orientation = "horizontal") #3a
        label1 = Label(text="hello") #3b1
        btn2 = Button(text="button") #3b2
        layout.add_widget(label1) #3c1
        layout.add_widget(btn2)  #3c2
        return layout

#4
HelloApp().run()
```

1. ウィンドウサイズの設定(#1の部分)
   1から３行の部分はGUIのウィンドウの大きさを設定する．現時点でMac(Anaconda由来のpython)ではデフォルトでは，フルスクリーンでGUIが作成されるので，Falseでフルスクリーンとしない．
   さらに，ウィンドウのサイズはWindow.siz(widht,hight)で設定できる(単位はピクセル)．とりあえず500,300で設定．
2. モジュールのimport(#2)
   Kivyを利用するときにはkivyパッケージからappをimportしておく（kivyのアプリケーションを起動するために必要なもの）
   パーツとしてLabelを利用するのでLabelもimportしておく（他に必要なパーツの種類があるときには，それぞれをimportしておく）
   今回はさらにレイアウト(パーツの並べ方)のためのboxlayoutと，Buttonパーツのためのを導入する．
   レイアウトの種類について
   [Buttonについて ]https://kivy.org/doc/stable/api-kivy.uix.button.html)の公式を参照．
3. クラスの設計(#3)
   クラスを作成し(#3)，そのクラスを実行する(#4)という形でGUIを作成する．
   クラスの引数はAppとする（これはkivy.appからimport)されている．クラスの中の初期関数（コンストラクタ）は，基本的に，selfを引数として，内部にパーツを設定していく
   そして関数の最初でレイアウトを決めるためのBoxlayout型のオブジェクトを作る．ここではBoxlayoutの引数として，パーツの並べ方の向きが横(horizontal)を設定しておく．

次にラベルパーツとしてlabe1，そしてButtonクラスでbtn2というオブジェクトを作成する．
この2つのパーツをlayoutオブジェクトにくっつけていくためにadd_widgedtメソッドを用いる(3c1と3c2)
そして作成したlayoutというオブジェクトを戻り値とする．

4. クラスの実行(#4)
上記で作成したクラスHelloAppを実行させる．このとき実行のメソッドとしてrun()を利用する．

レイアウトの中にレイアウトを付け加えるといったことも可能なため，上記の方針である程度GUI画面を作成することができる．

## kvファイルを用いたレイアウトの設定

上記のようなレイアウトの仕方に対して，kivyではレイアウトを設定するkvファイルとpyファイルを分離して書くことができる．

まず，Pythonファイルは以下のように作成する(main.py)．

```
from kivy.core.window import Window #1
Window.fullscreen = False
Window.size = (500,300)

from kivy.app import App #2

#3
class TestApp(App):
    pass

#4
if __name__ == '__main__':
    TestApp().run()
```

ここまでと異なるのは#3の部分である．classのTestApp部分の処理内容はpassと表記し，以下のkivyファイルに画面の設定をさせる．なお#4の「if **name** =="**main**"」の条件式の部分は，このファイルがmainとして呼び出されたときには，以下の実行をするという意味であり，TestAppクラスを実行するようになっている．

上記のPythonファイルと同一ディレクトリに以下のkivyファイルを作成しておく(test.kv)．

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

kivyファイルでは

> パーツ種類:
> __パーツの要素: 要素内容の指定

という流れで記載しておく．(__の部分は実際にはtabでインデントする)
以下のようなkivyファイル(test.kv)に書き変えると複数のパーツをレイアウトすることができる

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

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

kivyファイルでレイアウトを扱うときには，以下のよう階層構造を示してやる．

> レイアウト種類:
> __パーツ種類:
> __ __ パーツの要素:要素内容の指定

## kivy言語を用いた書き方

上記のようにpythonファイルとkivyファイルを分けずに書く場合には，Builderをimportし，pythonファイルの内部に書き込んでいく．


greeting
hoge
朝
昼
夜

```
from kivy.core.window import Window
Window.fullscreen = False
Window.size = (500,300)

from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import StringProperty, ListProperty
import japanize_kivy
from kivy.lang import Builder #1

#2
Builder.load_string('''
<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

         Button:
            id: button2
            text: "昼"
            font_size: 68

         Button:
            id: button3
            text: "夜"
            font_size: 68
''')

#3
class TextWidget(Widget):
   text  = "hoge"
   color = ListProperty([1,1,1,1])

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

```
        self.text = 'hoge'


class TestApp(App):
    def build(self):
        self.title = 'greeting'
        return TextWidget()

TestApp().run()
```

1. Builderのimport(#1)
   Pythonファイルに書き込むためにBuilderをimportする
   また日本語を用いいるためにパッケージもimportしておく.
2. kivyの記入(#2)
   kivy言語を記入する前後に以下のように記入する.

```
Builder.load_string('''
:
kivyの記入
''')
```

このとき<クラス名>でパーツ用のクラスの名前を記載しておく

3. パーツ用のクラスの設計(#3)
   上記で指定したクラス名のクラスをつくっていく．引数はwidgetとする.

# イベント処理

次にイベント処理（ボタンが押されると，何かしらの挙動が開始したら，テキストが変更されるなど）を説明する.
以下のボタンを押すと挨拶が表示されるGUIを作成してみる


greeting
朝
昼
夜


greeting
おはよう
朝
昼
夜

```
from kivy.core.window import Window
Window.fullscreen = False
Window.size = (500,300)

from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import StringProperty, ListProperty
import japanize_kivy
from kivy.lang import Builder

Builder.load_string('''
<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()  # ボタンをクリックした時

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

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

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

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

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

    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 build(self):
        self.title = 'greeting'
        return TextWidget()

TestApp().run()
```

上記のkivyファイルをまとめたものを変更したパターンとなっている.
このときkivy言語を書き込むところに，on_pressという記述があることに注目.
ここにクリックされたときに挙動する関数の名前をしてし，下のパーツ用のクラスの中に，その関数を作ることで挙動させることができる.

以下，2つほど実際に挙動するパターンとして
まずはクリックしたカウントをするアプリ

```
from kivy.core.window import Window
Window.fullscreen = False
Window.size = (500,300)

from kivy.lang import Builder
from kivy.uix.widget import Widget
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.properties import BooleanProperty
from kivy.properties import NumericProperty
from kivy.clock import Clock
from kivy.app import App


Builder.load_string('''
#:kivy 1.8.0

<KivyTimer>:
    BoxLayout:
        orientation: 'vertical'
        pos: root.pos
        size: root.size

        Label:
            text: str(root.left_time)
            font_size: 100

        BoxLayout:
            orientation: 'horizontal'
            size_hint: 1.0, 0.3

            Button:
                text: 'push'
                font_size: 16
                on_press: root.on_command('push')

            Button:
                text: 'Reset'
                font_size: 16
                on_press: root.on_command('reset')
''')


class KivyTimer(Widget):
    is_countdown = BooleanProperty(False)
    left_time = NumericProperty(0)

    def on_command(self, command):
        if command == 'push':
            self.left_time += 1
```

```
        elif command == 'reset':
            self.stop_timer()
            self.left_time = 0

    def on_countdown(self, dt):
        self.left_time -= 1
        if self.left_time == 0:
            self.is_countdown = False
            return False

    def start_timer(self):
        self.is_countdown = True
        Clock.schedule_interval(self.on_countdown, 1.0)
        pass

    def stop_timer(self):
        self.is_countdown = False
        Clock.unschedule(self.on_countdown)
        pass


class KivyTimerApp(App):
    def build(self):
        return KivyTimer()


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

次に時間を計測するアプリとして

```
from kivy.core.window import Window #1
Window.fullscreen = False
Window.size = (500,300)

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.clock import Clock
from kivy.properties import NumericProperty

class MyLabel(Label):
    time =NumericProperty(0)
    def on_time(self,*args):
        self.text =str(self.time)

class MyButton(Button):
    env = None
    def on_press(self):
        if self.text == "start":
            self.evt = Clock.schedule_interval(self.cb,0.2)
            self.text = "stop"
        else:
            self.evt.cancel()
            self.text ="start"

    def cb(self,dt):
        self.parent.lbl.time = round(self.parent.lbl.time + 0.2, 1)

class StopWatchApp(App):
    def build(self):
        layout = BoxLayout(orientation="horizontal")
        layout.lbl=MyLabel(text="0")
        layout.btn = MyButton(text="start")
        layout.add_widget(layout.lbl)
        layout.add_widget(layout.btn)
        return layout

StopWatchApp().run()
```

を示しておく.

## kivyのその他の設定

以下，画像を表示させる方法と画面遷移の方法について簡単に示す
期末課題などで挑戦したい方はぜひ.

## 画像の表示

```
from kivy.core.window import Window
Window.fullscreen = False
Window.size = (500,300)

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


class MyApp(App):
    def build(self):
        hoge = Image(source="img/fig1.png")
        return hoge

MyApp().run()
```

これまでのLabelやButtonと同様に画像もImageというクラスでオブジェクトを作成する．注意する点としては，Imageクラスの引数はsource="画像ファイルのディレクトとファイル名"と指定すること

参照先

## 画面遷移

```
from kivy.core.window import Window
Window.fullscreen = False
Window.size = (500,300)

from kivy.app import App
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager, Screen

# Create both screens. Please note the root.manager.current: this is how
# you can control the ScreenManager from kv. Each screen has by default a
# property manager that gives you the instance of the ScreenManager used.
Builder.load_string("""
<MenuScreen>:
    BoxLayout:
        Button:
            text: 'Goto settings'
            on_press: root.manager.current = 'settings'
        Button:
            text: 'Quit'

<SettingsScreen>:
    BoxLayout:
        Button:
            text: 'My settings button'
        Button:
            text: 'Back to menu'
            on_press: root.manager.current = 'menu'
""")

# Declare both screens
class MenuScreen(Screen):
    pass

class SettingsScreen(Screen):
    pass

# Create the screen manager
sm = ScreenManager()
sm.add_widget(MenuScreen(name='menu'))
sm.add_widget(SettingsScreen(name='settings'))

class TestApp(App):

    def build(self):
        return sm

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

参照先

# 提出物

- **レビューシート** : 今回の講義の感想.
- **演習課題** : 別途資料を参考に提出（〆切は次回講義前日の23:59まで)