SIT32004
ICT Application Development

Introduction to Kivy

Prof. Changbeom Choi

Supported by Jimin Jeong

Kivy

- **What is Kivy?**
  - Kivy emerges as a successor of PyMT (a library for multitouch applications)
  - Able to use **same code for cross-platform**:
    - » Linux / Windows / Mac OS X / Androids / iOS / Raspberry Pi
  - **Python framework** designed for developing **natural user interfaces**.
  - Possible to build a Kivy application using **pure Python code** and **Kivy as a library** which allow us to modify interfaces dynamically.

Introduction to Kivy

- Kivy contains all the basics of building a Graphical User Interface (GUI)
  - The following is a list of all the skills that you're about to learn:
    - » **Launching a Kivy application**
    - » The Kivy language
    - » Creating and using **widgets (GUI components)**
    - » **Basic properties** and **variables** of the widgets
    - » Fixed, proportional, absolute, and relative **coordinates**
    - » Organizing GUIs through **layouts**
    - » Tips for achieving **responsive GUIs**

- Apart from Python, this lecture requires some knowledge **about Object-Oriented Programming concepts.**
- In particular, **inheritance** and the **difference between instances and classes** will be assumed.

Practice01: App, Label

- Two classes from the Kivy library: **App and Label.**
  - The **App class** is the **starting point of any Kivy application** and the way we use the App class is through **inheritance.**
  - App becomes the **base class** of HelloApp, the subclass or child class.
  - The HelloApp's body just modifies the **build(self) method.** This method returns **the window content.**

```
#File name: practice1.py
import kivy
from kivy.app import App
from kivy.uix.button import Label

class HelloApp(App):
    def build(self):
        return Label(text='Hello World!')

if __name__=="__main__":
    HelloApp().run()
```

Practice02: Executing Kivy Application (1/2)

- Open **IDLE** (Python Development Tool)

Python 2.7.15 Shell

File Edit Shell Debug Options Window Help

```
Python 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:30:26) [MSC v.1500 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

practice1.py - C:/Users/Jenny/Desktop/practice1.py (3.6.8)

File Edit Format Run Options Window Help

```
#File name: practice1.py
import kivy
kivy.require('1.7.0')

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

class HelloApp(App):
    def build(self):
        return Label(text='Hello World!')

if __name__=="__main__":
    HelloApp().run()
```

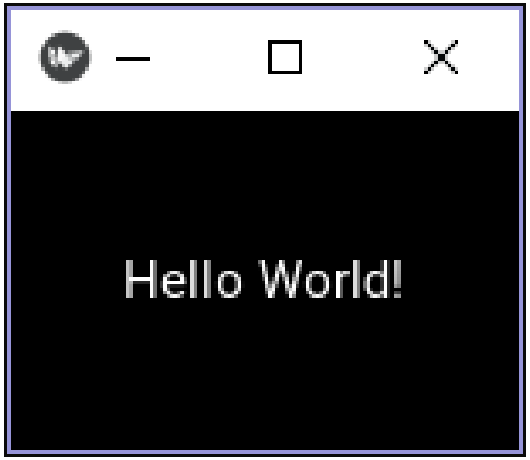
- Ctrl + N to make file
- Copy the previous code
- Save it as practice1.py
- Open **Command line (Window+R, type cmd)**
- Go to where practice1.py is saved
- Type **python practice1.py** or **python practice1.py --size=150x100** (to specify the screen size)

Practice02: Executing Kivy Application (2/2)

- **Command Line Example**

```
C:₩Users₩Jenny₩Desktop>python practice1.py --size=150x100
```

- **Result**



Practice02: Separation of Concern

- Separation of concerns (SoC) is a design principle for separating a computer program into distinct sections, so that each section addresses a separate concern.
  - In our application, we separate application logic and designs
  - *.py files contains the application logic
  - *.kv files contains the application design (UI considerations)

```python
# File name: practice2.py
from kivy.app import App
from kivy.uix.button import Label

class practice2App(App):
    def build(self):
        return Label()

if __name__=="__main__":
    practice2App().run()
```

```kv
#File name: practice2.kv

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

Practice03: Ping Pong Game (1/6)

- Create a directory for the game and a file named main.py
  - Following code creates an instance of our PongGame Widget class and returns it as the root element for the applications UI.

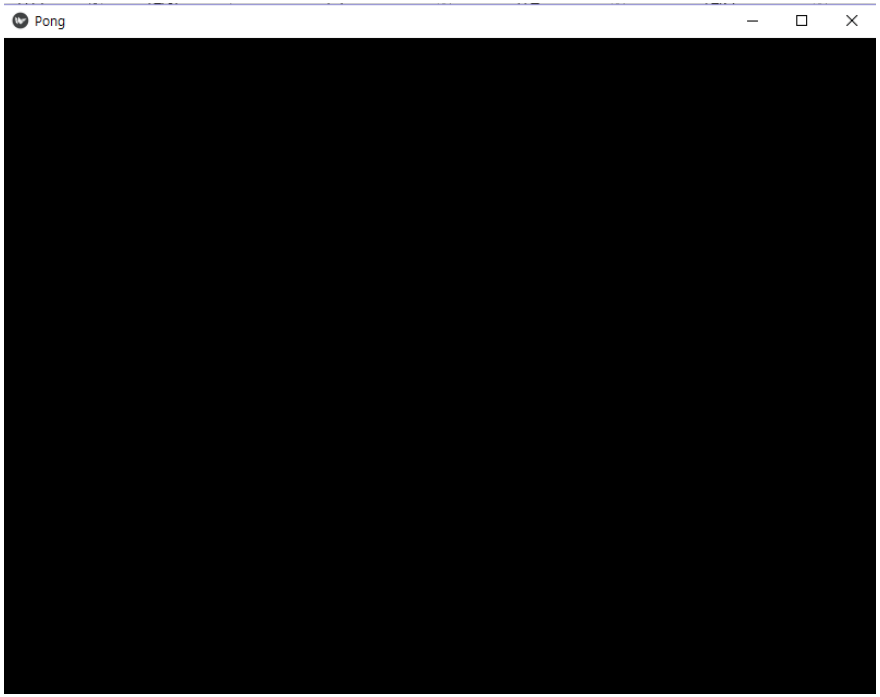
```
#File name: main.py

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

class PongGame(Widget):
    pass

class PongApp(App):
    def build(self):
        return PongGame()

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



Practice03: Ping Pong Game (2/6)

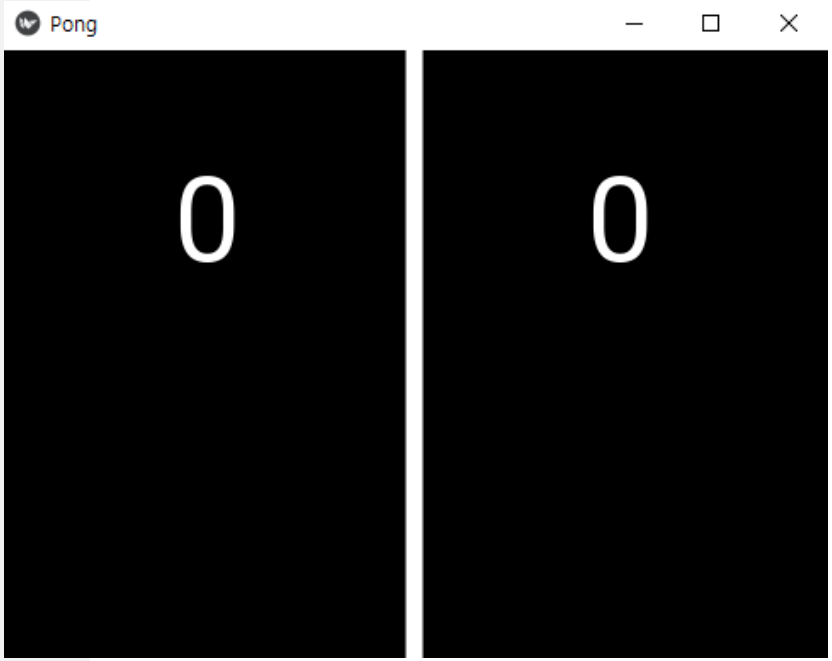
- Create pong.kv and add the following code.
  - A block defined with a class name inside the < > characters is a **Widget rule**.

```
#File name: pong.kv

<PongGame>:
    canvas:
        Rectangle:
            pos: self.center_x - 5, 0
            size: 10, self.height

    Label:
        font_size: 70
        center_x: root.width / 4
        top: root.top - 50
        text: "0"

    Label:
        font_size: 70
        center_x: root.width * 3 / 4
        top: root.top - 50
        text: "0"
```



Practice03: Ping Pong Game (3/6)

- Add a PongBall class to create a ball widget and make it bounce around.

```
#File name: main.py

from kivy.properties import NumericProperty, ReferenceListProperty
from kivy.vector import Vector


class PongBall(Widget):
    velocity_x = NumericProperty(0)
    velocity_y = NumericProperty(0)

    velocity = ReferenceListProperty(velocity_x, velocity_y)

    def move(self):
        self.pos = Vector(*self.velocity) + self.pos
```

ReferenceListProperty allows to use ball.velocity as a shorthand

“move” function will move the ball one step.
This will be called in equal intervals to animate the ball.

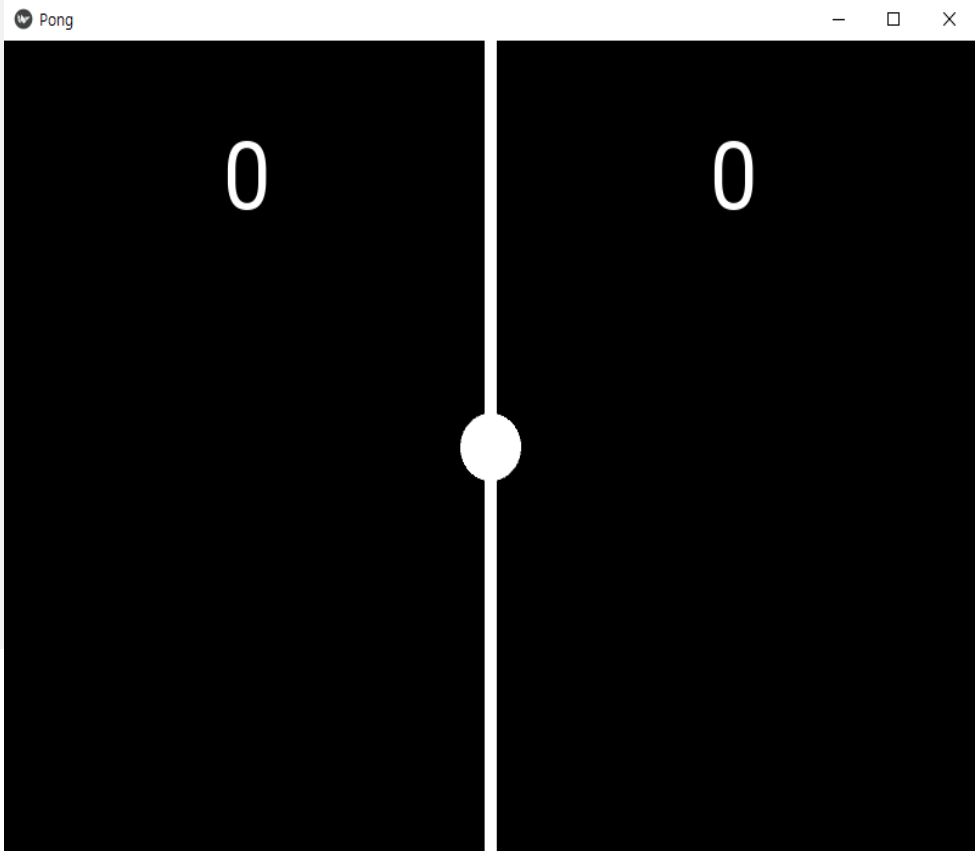
Practice03: Ping Pong Game (4/6)

- Add the ball
  - The kv rule used to draw the ball as a white circle

```
#File name: pong.kv

<PongBall>:
    size: 50, 50
    canvas:
        Ellipse:
            pos: self.pos
            size: self.size


<PongGame>
    ...
    PongBall:
        center: self.parent.center
```



Practice03: Ping Pong Game (5/6)

- **Adding ball animation**
  - We need the move method of our ball to be called regularly. In this case, use **Clock** function.

```
#File name: main.py
from kivy.clock import Clock
Clock.schedule_interval(game.update, 1.0 / 140.0)
```

```
#File name: main.py

class PongGame(Widget):
    ball = ObjectProperty(None)

  def serve_ball(self):
      self.ball.center = self.center
      self.ball.velocity = Vector(4, 0).rotate(randint(0, 360))

   def update(self, dt):
       pass
```

With **ObjectProperty**, ball can easily reference the ball property inside the update method.

Practice03: Ping Pong Game (6/6)

- Adding Interval scheduling features to the PongApp

```
#File name: main.py

class PongApp(App):

    def build(self):
        game = PongGame()
        game.serve_ball()
        Clock.schedule_interval(game.update, 1.0 / 140.0)
        return game
```

- We need the movable player rackets and keeping track of the score.

```
#File name: main.py

def on_touch_move(self, touch):
    if touch.x < self.width/3:
        self.player1.center_y = touch.y
    if touch.x > self.width - self.width/3:
        self.player2.center_y = touch.y
```

Set the position of the left or right player based on whether the touch occurred on the left or right side of the screen with **on_touch_move** handler.

Homework04

- Put additional code in update() handler to bouce paddles and ball, and to score point if the ball went of to a side.
  - Game should be successfully played.
  - You should put comments(#) at every code you put!!
    Explain your code (You can write in Korean) and it must be reasonable.