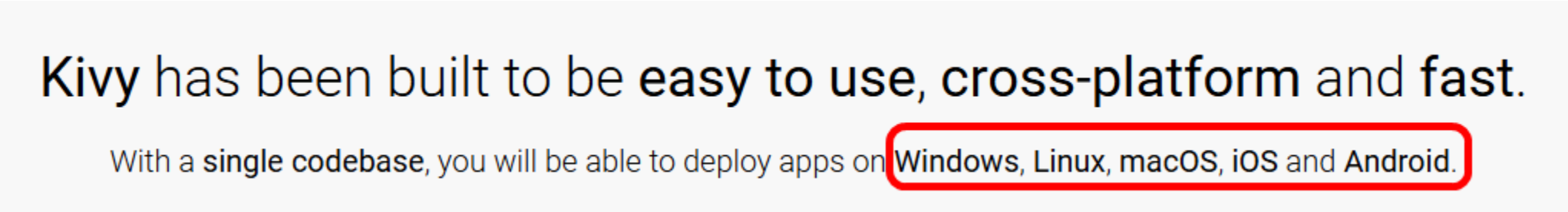
# Kivy

賴璉錡
lclai.t11@o365.fcu.edu.tw

# kivy - The Open Source Python App Development Framework.

- https://kivy.org/



# Kivy - 基本架構

```
from kivy.app import App


class Example1App(App):  1 usage
    def build(self):
        return None


if __name__ == '__main__':
    Example1App().run()
```

這行是在「引入」(import) 我們需要的模組

build 是一個特殊的函式名稱，它是 Kivy 應用程式的核心。這個函式會決定應用程式啟動時要顯示什麼內容

定義一個新的「應用程式類別」，(App) 表示這個程式是以 Kivy 的 App 為基礎來建立的，繼承了 App 的所有基本功能

self 代表類別本身，讓程式能夠參考到自己這個類別，能使用本身的參數與方法

它的意思是「如果這個程式是直接被執行的話」

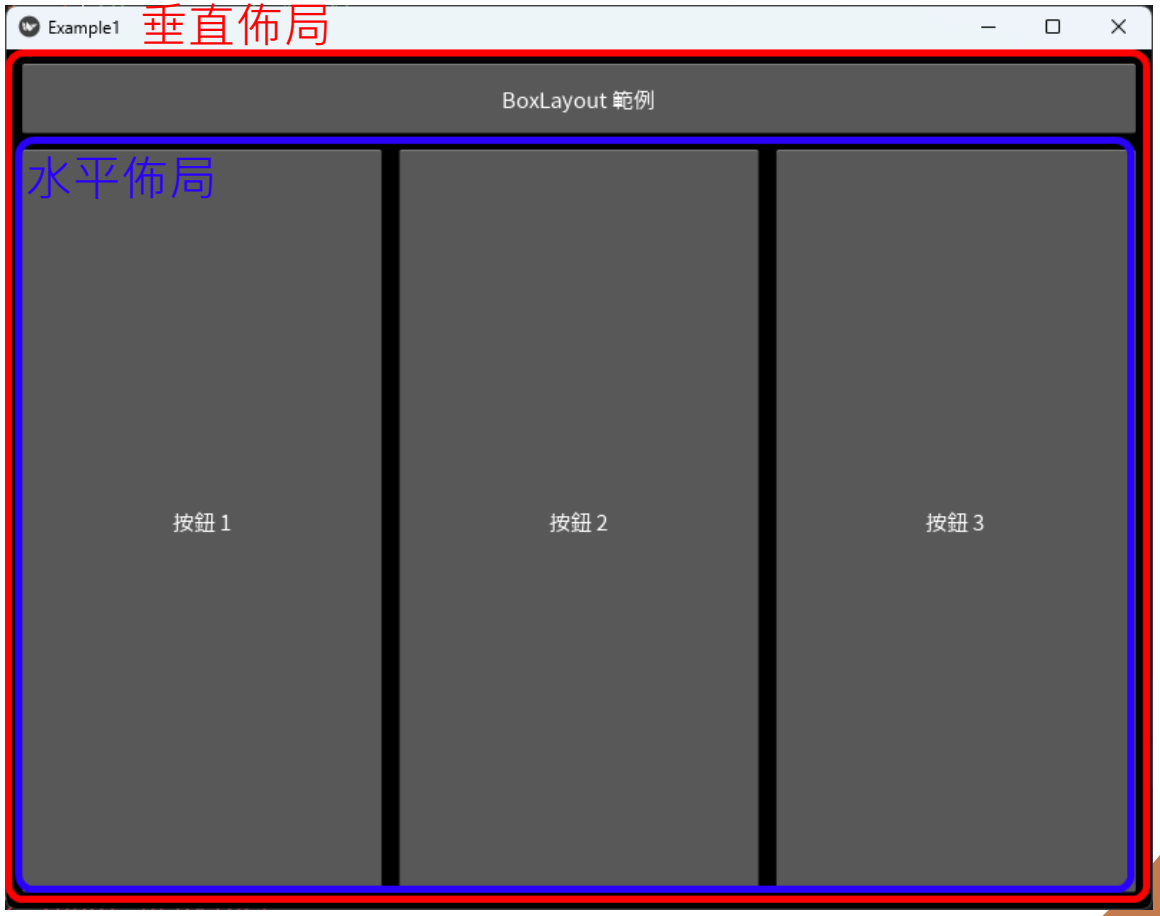
這行是真正啟動應用程式的指令

- Example1App() 創建一個新的應用程式實例
- .run() 是告訴程式「開始運行」

# Kivy - 基本佈局

- BoxLayout:垂直與水平排列

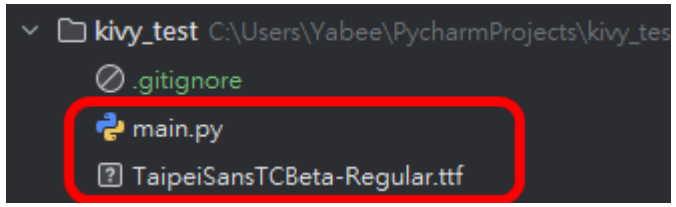
```
# 複製以下程式碼
class Example1App(App):
    def build(self):
        # 外層垂直佈局
        root = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # 加入一個標題按鈕
        root.add_widget(Button(
            text='BoxLayout 範例',
            size_hint_y=None,
            height=50
        ))

        #加入一個水平佈局
        h_layout = BoxLayout(orientation='horizontal', spacing=10)
        for i in range(3):
            h_layout.add_widget(Button(text=f'按鈕{i + 1}'))

        root.add_widget(h_layout)
        return root
```



# 引入中文字型

- 下載字型檔
  - 中文字型檔連結
- 建立 main.py 檔案

```
# 複製以下程式碼
from kivy.core.text import LabelBase
from kivy.resources import resource_add_path
import os

# 引用字體檔案
resource_add_path(os.path.abspath('TaipeiSansTCBeta-Regular.ttf'))
# 將kivy預設的字體替換成指定的中文字體
LabelBase.register('Roboto', 'TaipeiSansTCBeta-Regular.ttf')
```




```
from kivy.core.text import LabelBase
from kivy.resources import resource_add_path
import os

# 引用字體檔案
resource_add_path(os.path.abspath('TaipeiSansTCBeta-Regular.ttf'))
# 將kivy預設的字體替換成指定的中文字體
LabelBase.register( name: 'Roboto',  fn_regular: 'TaipeiSansTCBeta-Regular.ttf')
```

# Kivy - 基本元件

- 按鈕與文字標籤

引入按鈕與標籤

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

class Example1App(App):
    def build(self):
        layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # 建立標籤
        label = Label(
            text='Hello Kivy!',
            font_size=30,
            color=(1, 0, 0, 1) # 紅色
        )

        # 建立按鈕
        button = Button(
            text='點擊我',
            size_hint=(None, None),
            size=(200, 50),
            pos_hint={'center_x': 0.5}
        )
        button.bind(on_press=self.on_button_press)

        layout.add_widget(label)
        layout.add_widget(button)
        return layout

    def on_button_press(self, instance):
        print('按鈕被點擊了！')
```

建立一個佈局

```
class Example1App(App):
    def build(self):
        layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # 建立標籤
        label = Label(
            text='Hello Kivy!',
            font_size=30,
            color=(1, 0, 0, 1)  # 紅色
        )

        # 建立按鈕
        button = Button(
            text='點擊我',
            size_hint=(None, None),
            size=(200, 50),
            pos_hint={'center_x': 0.5}
        )
        button.bind(on_press=self.on_button_press)

        layout.add_widget(label)
        layout.add_widget(button)
        return layout

    def on_button_press(self, instance):  1 usage
        print('按鈕被點擊了！')
```

加入「size_hint=(None, None),」這一行才能自定義按鈕尺寸

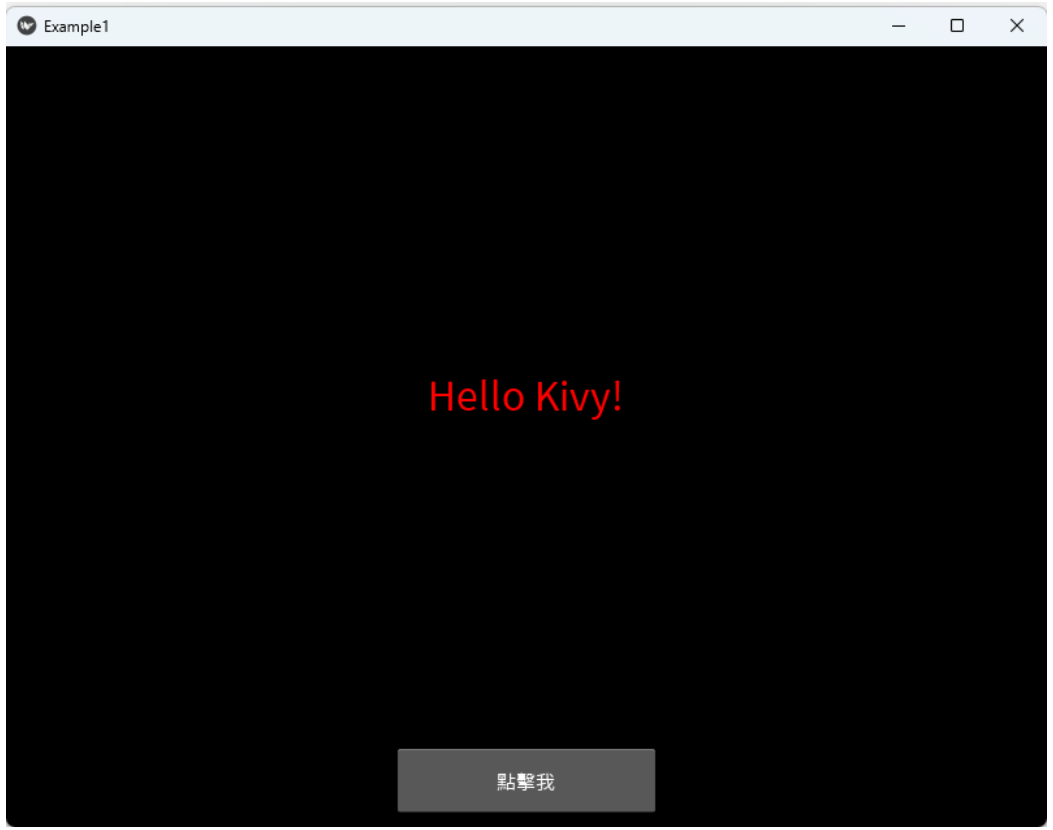
綁定「按下」這個動作到自
這個類別的「on_button_p
這個函式

加入這兩個元件到佈局之中

定義一個按鈕被按下的動作

# Kivy - 基本元件

- 按鈕與文字標籤




```
[INFO   ] [Window      ] auto add sdl2 input provider
[INFO   ] [Window      ] virtual keyboard not allowed, single mode, not docked
[INFO   ] [Base        ] Start application main loop
[INFO   ] [GL          ] NPOT texture support is available
按鈕被點擊了！
按鈕被點擊了！
按鈕被點擊了！
```

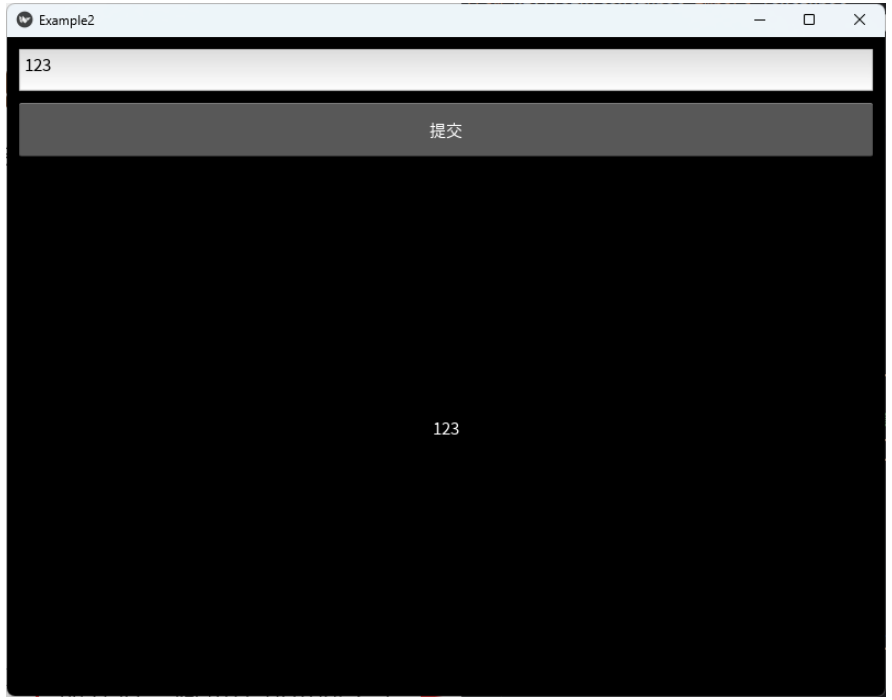
# Kivy - 基本元件

- 文字輸入和顯示

```
from kivy.uix.textinput import TextInput


class Example2App(App):
    def build(self):
        layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # 建立文字輸入框
        self.input = TextInput(
            multiline=False,
            hint_text='請輸入文字',
            size_hint_y=None,
            height=40
        )

        # 建立顯示標籤
        self.label = Label(text='輸入的文字將顯示在這裡')

        # 建立提交按鈕
        submit_btn = Button(
            text='提交',
            size_hint_y=None,
            height=50
        )
        submit_btn.bind(on_press=self.update_label)

        layout.add_widget(self.input)
        layout.add_widget(submit_btn)
        layout.add_widget(self.label)
        return layout

    def update_label(self, instance):
        self.label.text = self.input.text
```

# Kivy - 基本元件

- 文字輸入和顯示




```python
from kivy.uix.textinput import TextInput

class Example2App(App):
    def build(self):
        layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # 建立文字輸入框
        self.input = TextInput(
            multiline=False,
            hint_text='請輸入文字',
            size_hint_y=None,
            height=40
        )

        # 建立顯示標籤
        self.label = Label(text='輸入的文字將顯示在這裡')

        # 建立提交按鈕
        submit_btn = Button(
            text='提交',
            size_hint_y=None,
            height=50
        )
        submit_btn.bind(on_press=self.update_label)

        layout.add_widget(self.input)
        layout.add_widget(submit_btn)
        layout.add_widget(self.label)
        return layout

    def update_label(self, instance):  1 usage
        self.label.text = self.input.text
```

引入輸入框

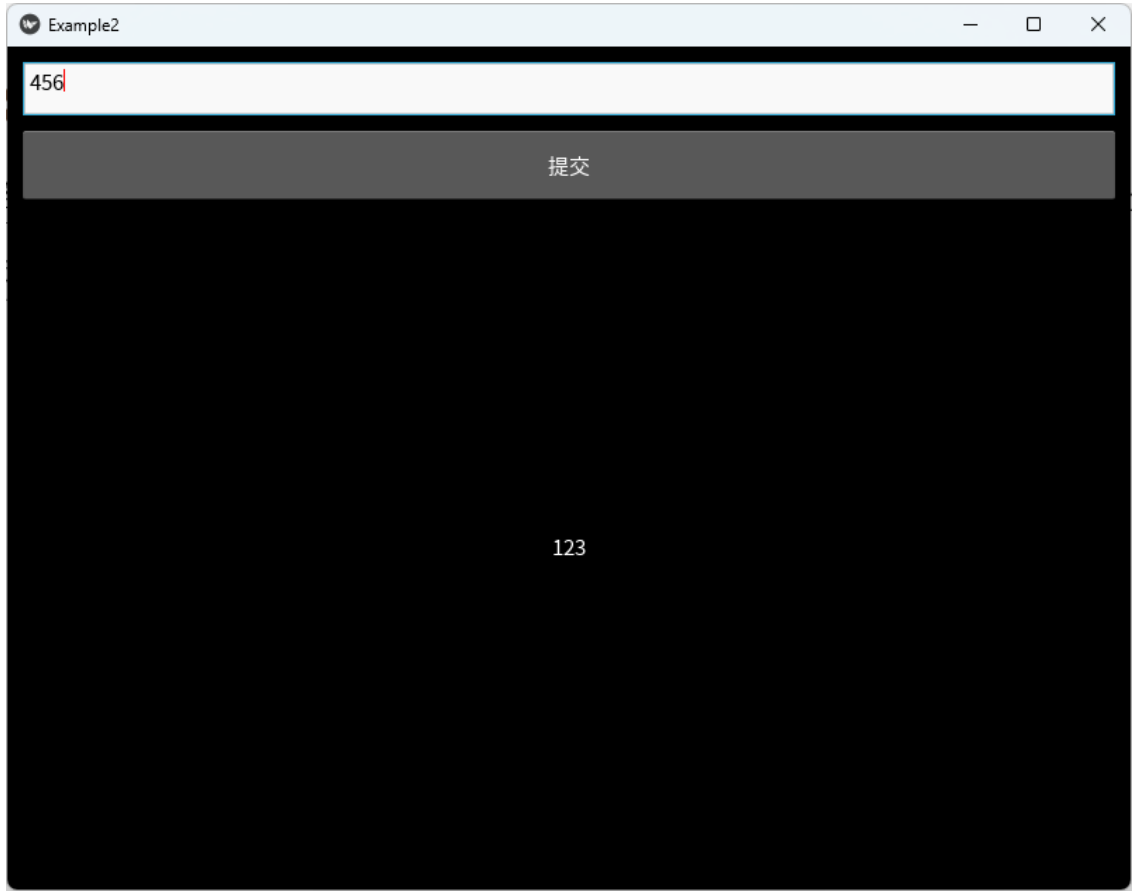
綁定「按下」這個動作到自己本
這個類別的「update_label」這
函式

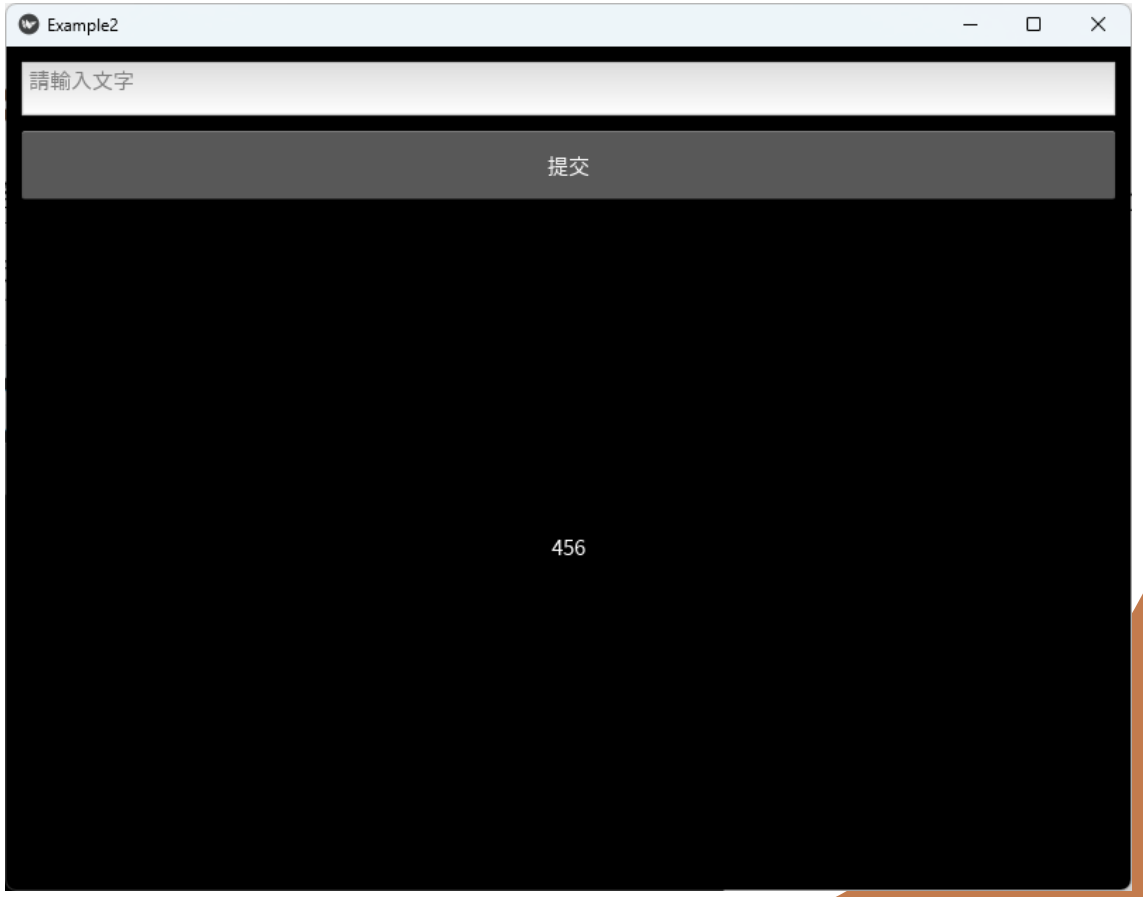
加入這三個元件到佈局之中

按下按鈕後更新輸入框的文
字到文字標籤

# Practice: 清除輸入框

- 請將Example2App這個測試輸入框的程式修改成更新文字標籤後，將輸入框的文字清除



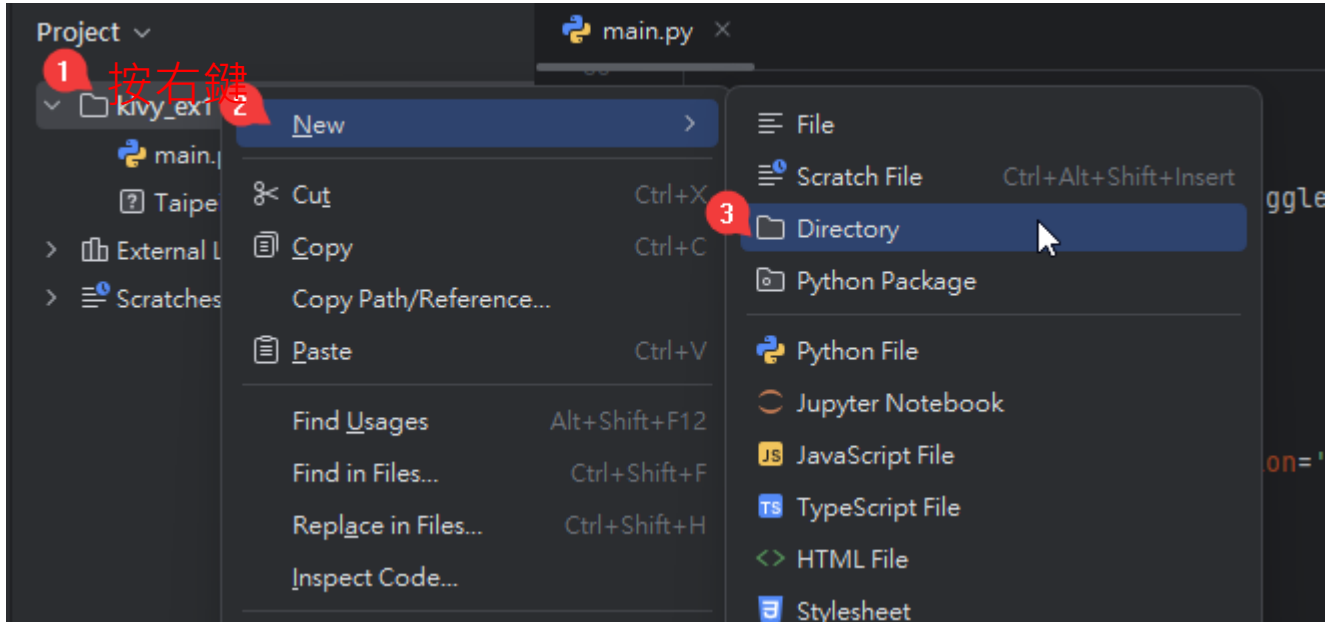
# Kivy - 基本元件

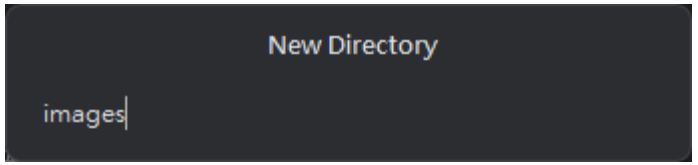
- 切換按鈕和圖片

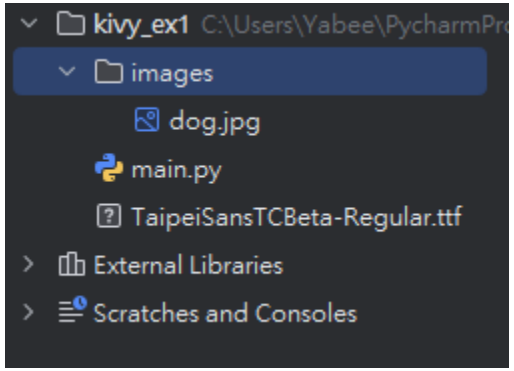
```
from kivy.uix.togglebutton import ToggleButton
from kivy.uix.image import Image


class Example3App(App):
    def build(self):
        layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # 建立切換按鈕
        toggle = ToggleButton(
            text='開關',
            size_hint=(None, None),
            size=(100, 50),
            pos_hint={'center_x': 0.5}
        )
        toggle.bind(on_press=self.on_toggle)

        # 建立圖片（使用一個簡單的形狀作為範例）
        self.img = Image(
            source='images/dog.jpg',  # 替換為實際圖片路徑
            size_hint=(None, None),
            size=(200, 200),
            pos_hint={'center_x': 0.5}
        )

        layout.add_widget(self.img)
        layout.add_widget(toggle)
        return layout

    def on_toggle(self, instance):
        if instance.state == 'down':
            self.img.source = 'images/dog.jpg'
            print("dog")
        else:
            self.img.source = 'images/dog2.jpg'
            print("dog2")
```

按鈕的狀態

# Kivy - 新增圖片


Project
main.py
1
按右鍵
kivy_ex1
2
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+Shift+F12
Find in Files... Ctrl+Shift+F
Replace in Files... Ctrl+Shift+H
Inspect Code...
3
File
Scratch File Ctrl+Alt+Shift+Insert
Directory
Python Package
Python File
Jupyter Notebook
JavaScript File
TypeScript File
HTML File
Stylesheet


New Directory
images

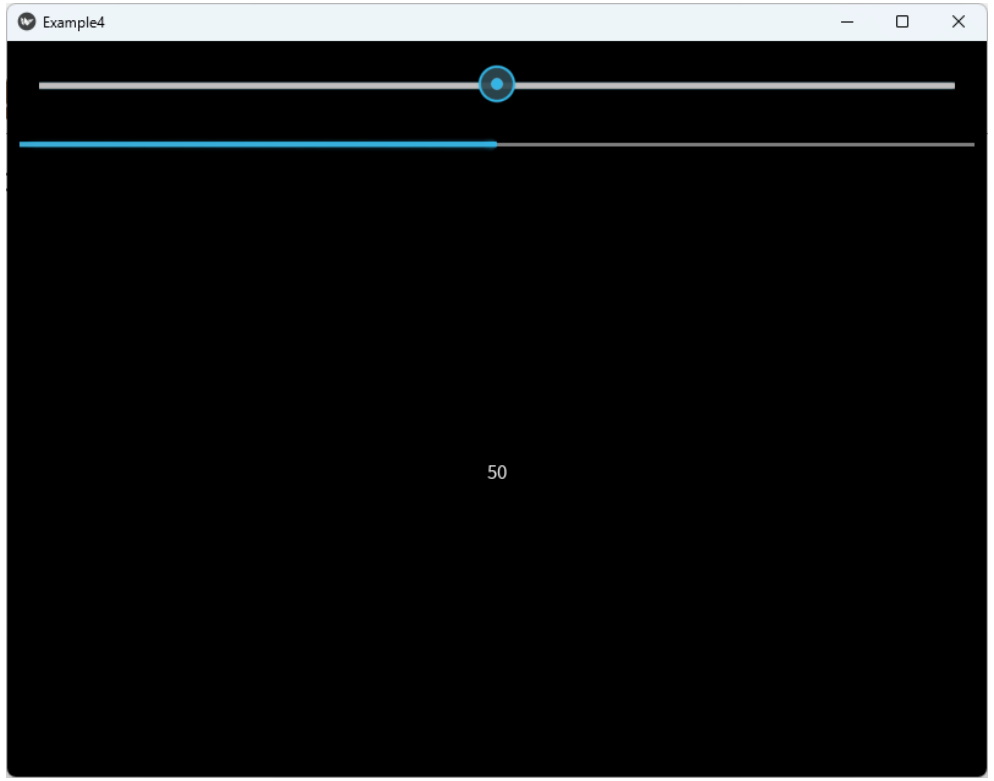

kivy_ex1 C:\Users\Yabee\PycharmPr
images
dog.jpg
main.py
TaipeiSansTCBeta-Regular.ttf
External Libraries
Scratches and Consoles

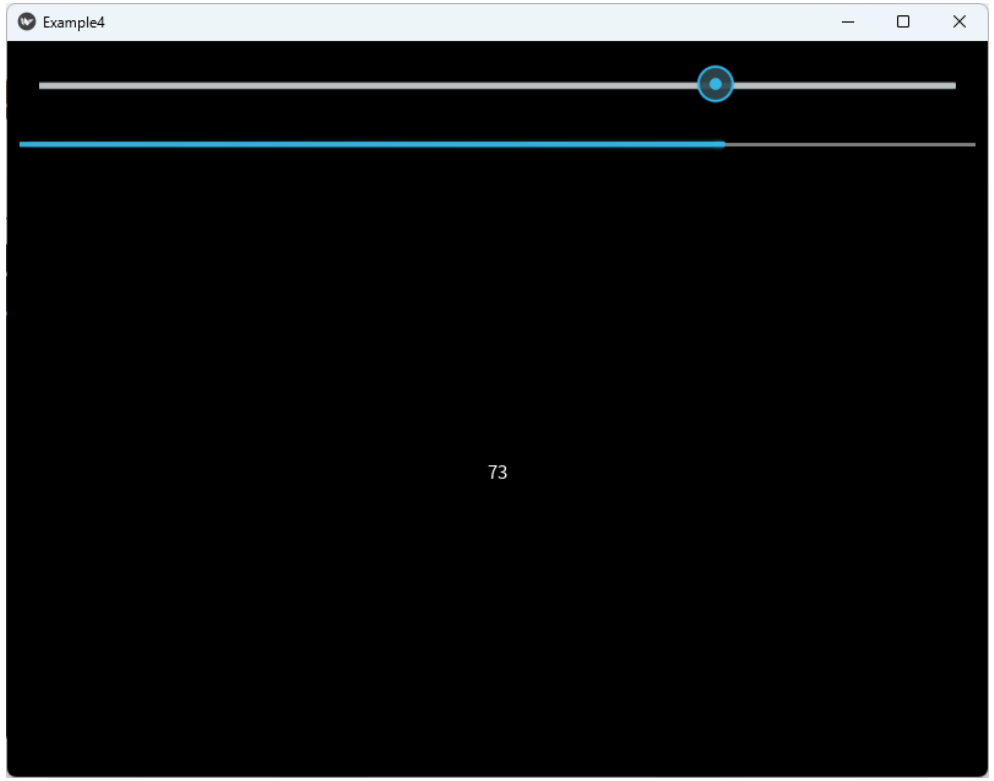
# Kivy - 基本元件

- 滑動條和進度條

```
from kivy.uix.slider import Slider
from kivy.uix.progressbar import ProgressBar


class Example4App(App):
    def build(self):
        layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # 建立滑動條
        slider = Slider(
            min=0,
            max=100,
            value=50,
            size_hint_y=None,
            height=50
        )
        slider.bind(value=self.on_slider_change)

        # 建立進度條
        self.progress = ProgressBar(
            max=100,
            value=50,
            size_hint_y=None,
            height=30
        )

        # 建立數值顯示標籤
        self.value_label = Label(text='50')

        layout.add_widget(slider)
        layout.add_widget(self.progress)
        layout.add_widget(self.value_label)
        return layout

    def on_slider_change(self, instance, value):
        self.progress.value = value
        self.value_label.text = str(int(value))
```

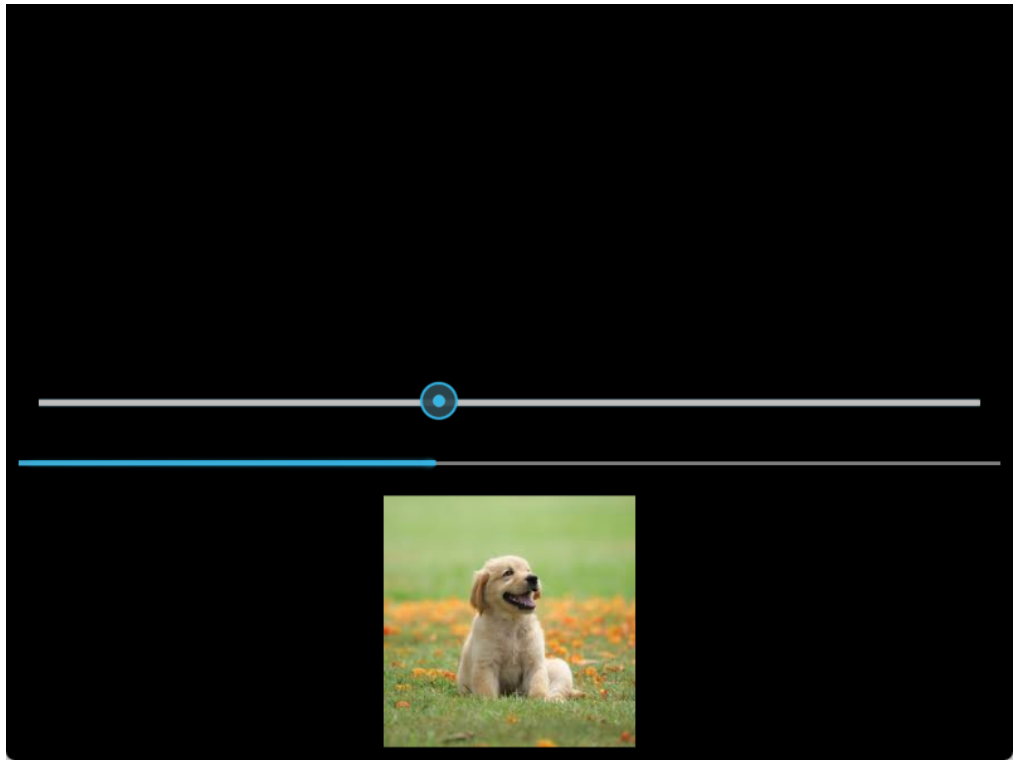
# Kivy - 滑動條和進度條

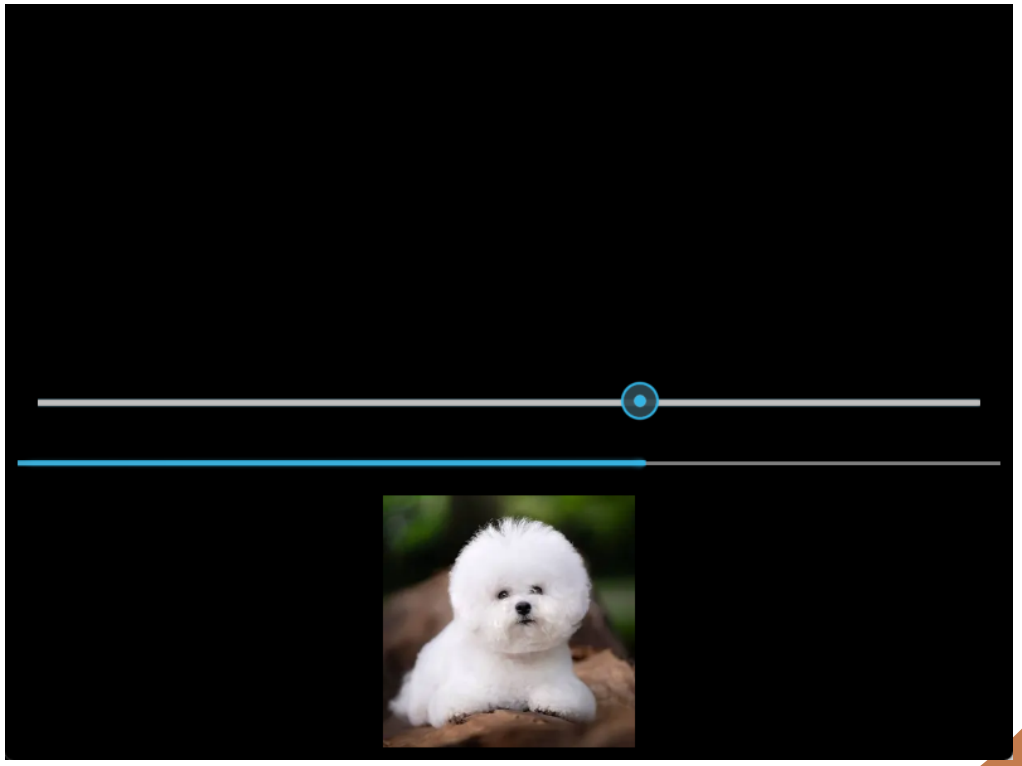


# Practice: 利用滑動進度條切換圖片

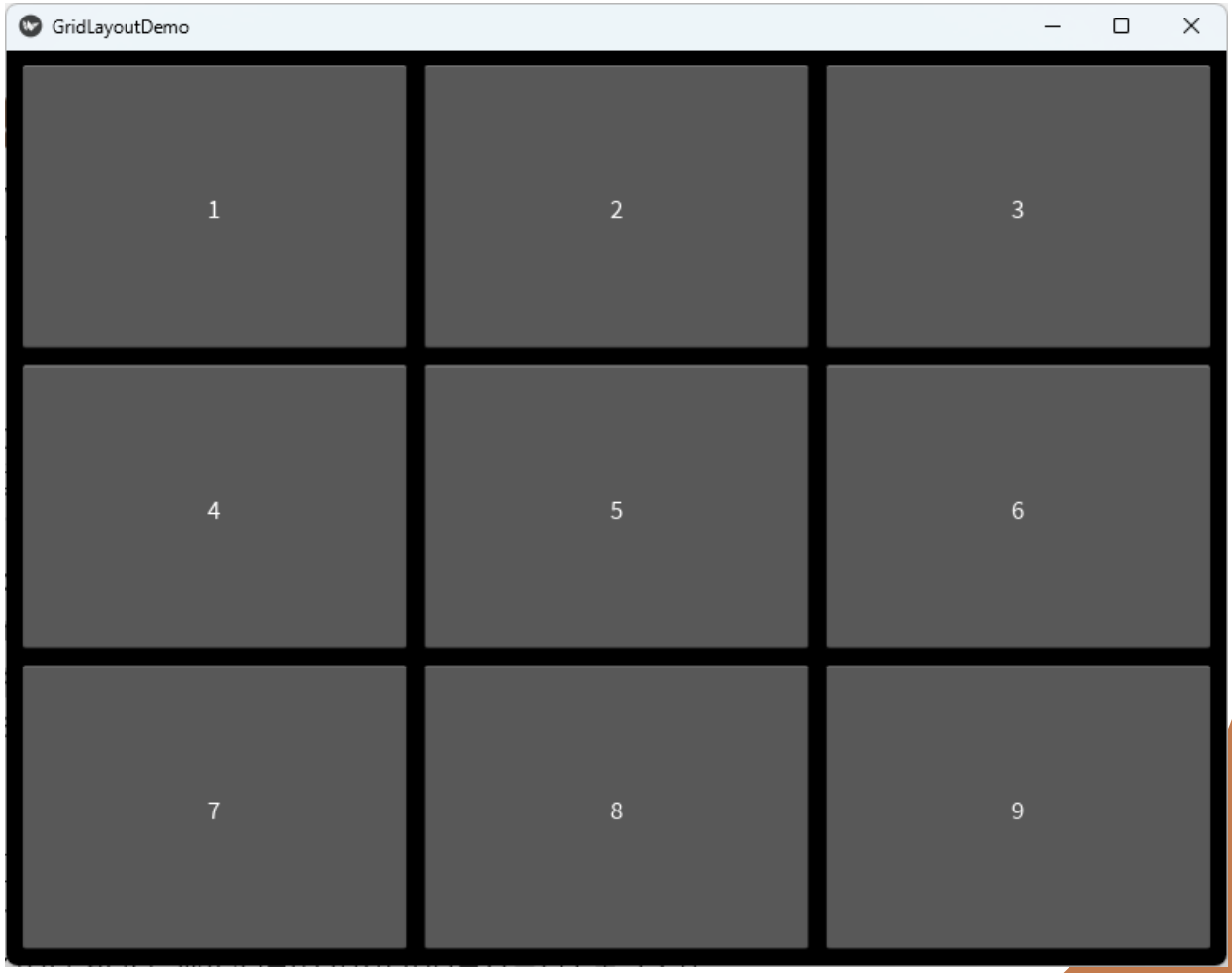
- 數值小於50顯示圖片1，大於50顯示圖片2





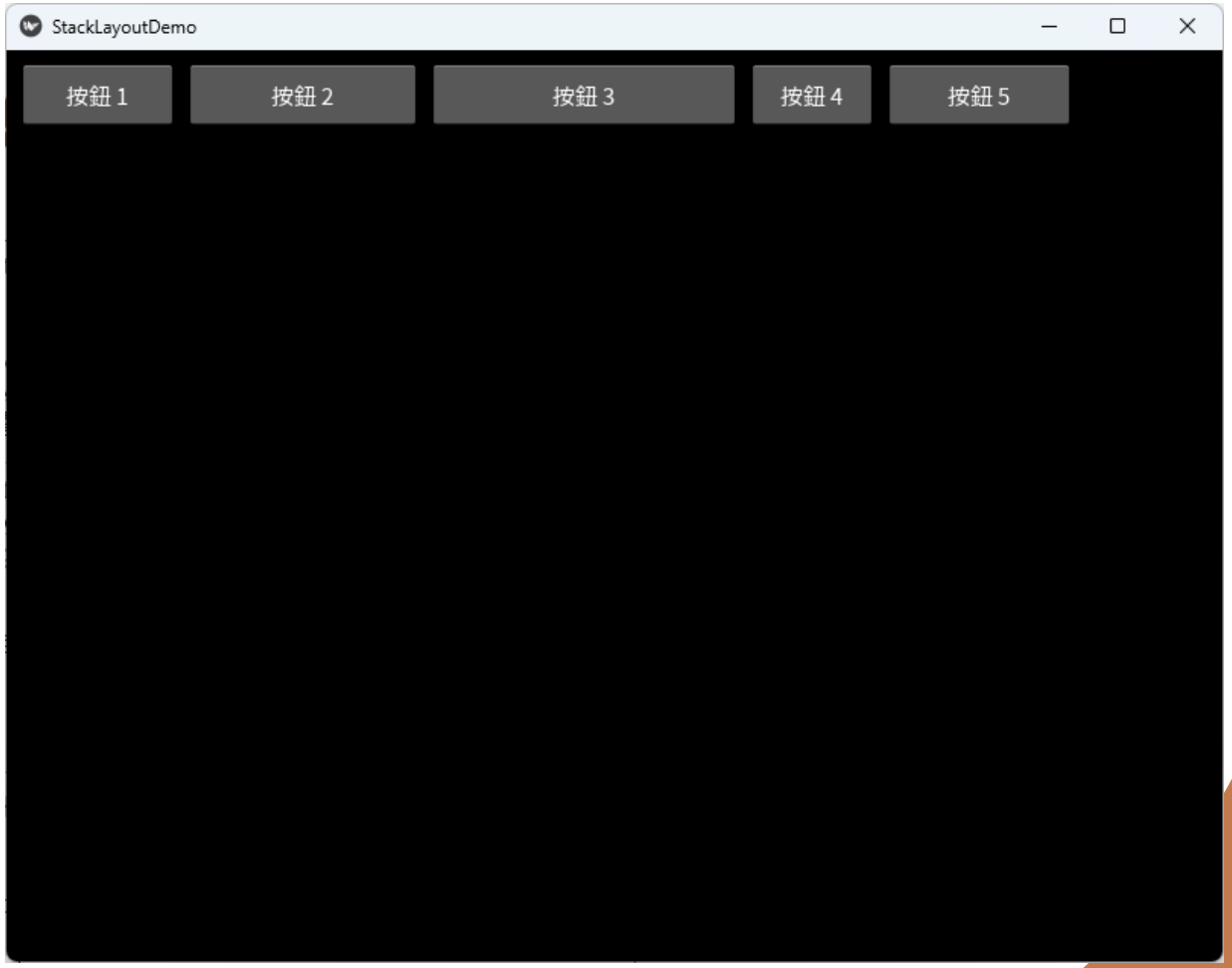
# Kivy – GridLayout 網格佈局

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


class GridLayoutDemo(App):
    def build(self):
        # 建立3x3網格
        grid = GridLayout(
            cols=3,
            rows=3,
            padding=10,
            spacing=10
        )

        # 加入9個按鈕
        for i in range(9):
            grid.add_widget(Button(text=f'{i + 1}'))

        return grid
```



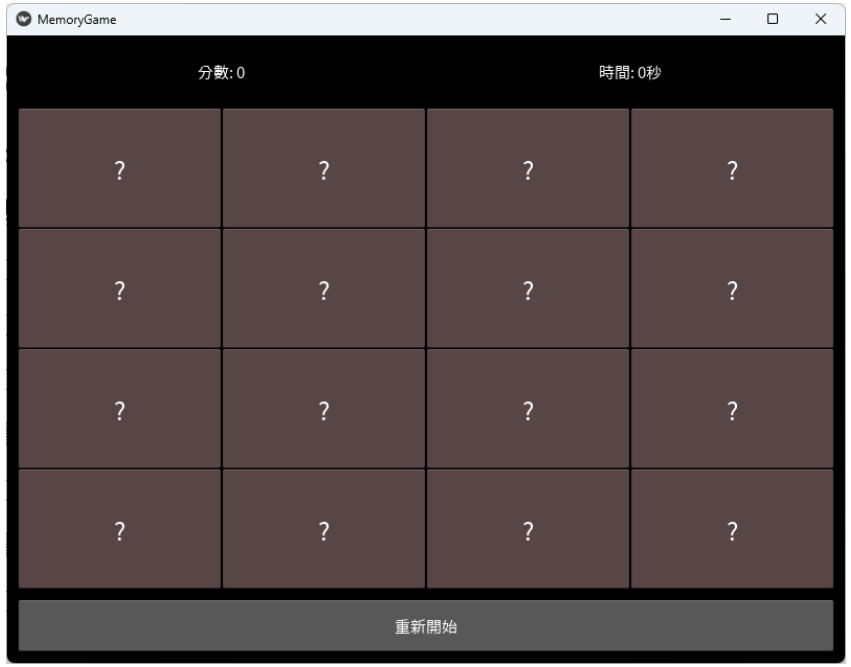
# Kivy – StackLayout 堆疊佈局

```
from kivy.uix.stacklayout import StackLayout


class StackLayoutDemo(App):
    def build(self):
        # 建立堆疊布局
        stack = StackLayout(
            orientation='lr-tb', # 從左到右，從上到下
            padding=10,
            spacing=10
        )

        # 加入不同大小的按鈕
        sizes = [(100, 40), (150, 40), (200, 40), (80, 40), (120, 40)]
        for i, size in enumerate(sizes):
            btn = Button(
                text=f'按鈕 {i + 1}',
                size_hint=(None, None),
                size=size
            )
            stack.add_widget(btn)

        return stack
```



# Practice: 翻牌配對遊戲

- Step:
  - 基本框架與布局
  - 加入資訊區域
  - 加入卡片網格
  - 加入卡片點擊功能
  - 實作配對檢查邏輯
  - 加入計分系統
  - 實作計時功能
  - 加入重新開始功能



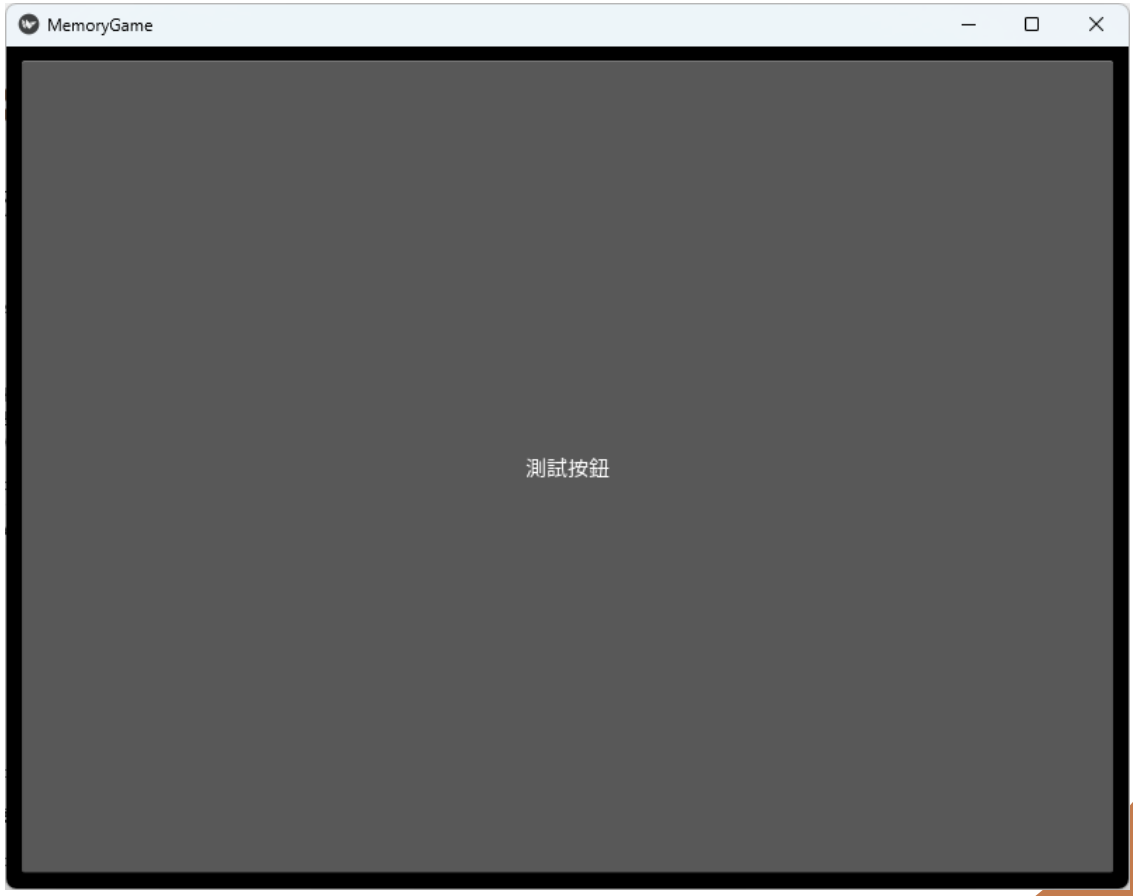

# Practice: 翻牌配對遊戲

- Step:
  - 基本框架與布局
  - 加入資訊區域
  - 加入卡片網格
  - 加入卡片點擊功能
  - 實作配對檢查邏輯
  - 加入計分系統
  - 實作計時功能
  - 加入重新開始功能

# Practice: 翻牌配對遊戲

- 基本框架與布局

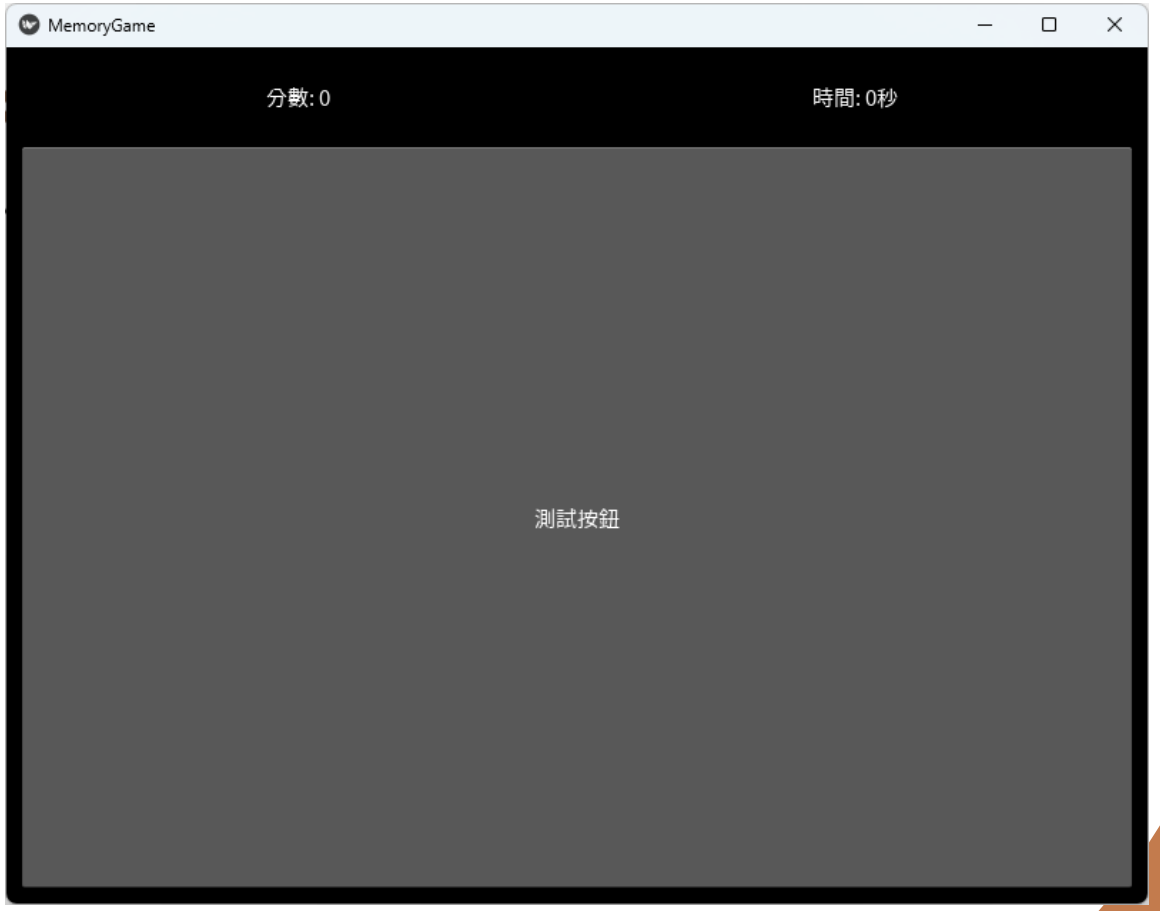
```
from kivy.core.text import LabelBase
from kivy.resources import resource_add_path
import os

# 引用字體檔案
resource_add_path(os.path.abspath('TaipeiSansTCBeta-Regular.ttf'))
# 將kivy預設的字體替換成指定的中文字體
LabelBase.register('Roboto', 'TaipeiSansTCBeta-Regular.ttf')

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.label import Label


class MemoryGame(App):
    def build(self):
        # 主布局
        self.main_layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # 遊戲網格
        self.game_grid = GridLayout(cols=4)

        # 簡單的測試按鈕
        test_button = Button(text='測試按鈕')
        self.game_grid.add_widget(test_button)

        self.main_layout.add_widget(self.game_grid)
        return self.main_layout


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



# Practice: 翻牌配對遊戲

- 加入資訊區域

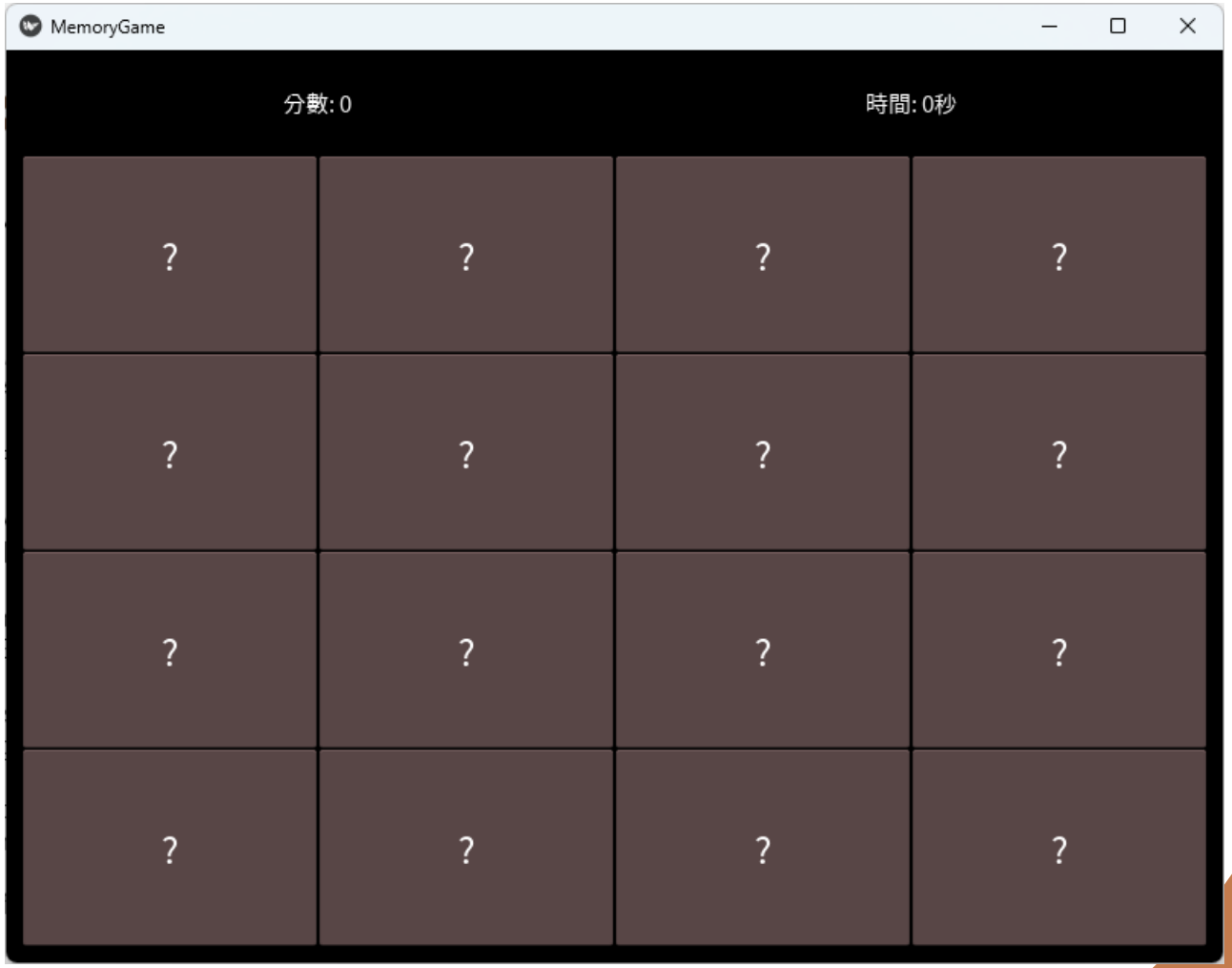
```
def build(self):
    self.main_layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

    # 加入資訊區域
    info_layout = BoxLayout(size_hint_y=None, height=50)
    self.score_label = Label(text='分數: 0')
    self.time_label = Label(text='時間: 0秒')
    info_layout.add_widget(self.score_label)
    info_layout.add_widget(self.time_label)

    self.game_grid = GridLayout(cols=4)
    test_button = Button(text='測試按鈕')
    self.game_grid.add_widget(test_button)

    self.main_layout.add_widget(info_layout)
    self.main_layout.add_widget(self.game_grid)
    return self.main_layout
```



# Practice: 翻牌配對遊戲

- 加入卡片網格

```
import random

def build(self):
    # ... 前面的布局程式碼 ...

    # 建立卡片
    self.cards = []
    self.card_values = list(range(8)) * 2
    random.shuffle(self.card_values)

    for i in range(16):
        card = Button(
            text='?',
            font_size=24,
            background_color=(1, 0.8, 0.8, 1)
        )
        self.cards.append(card)
        self.game_grid.add_widget(card)

    return self.main_layout
```



# Practice: 翻牌配對遊戲

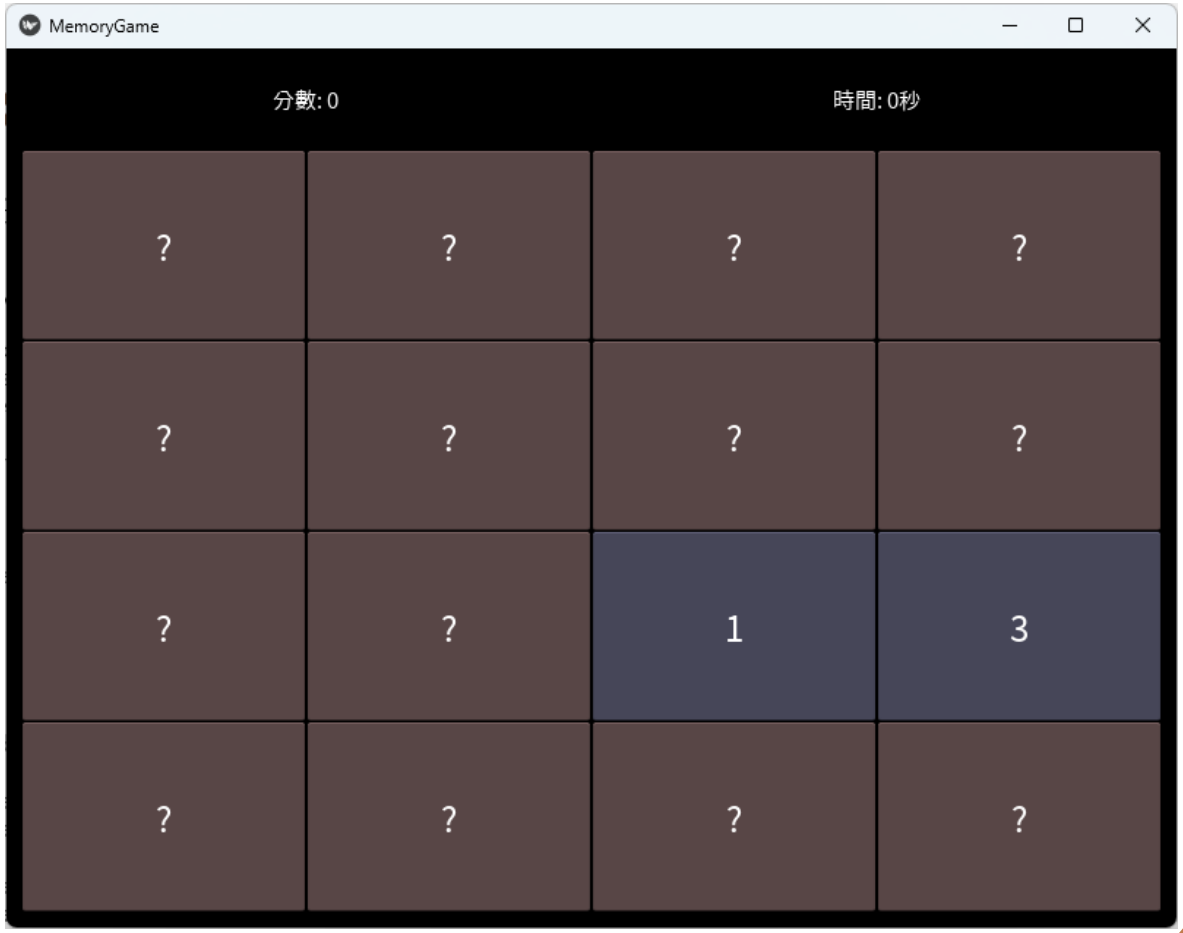
- 加入卡片點擊功能

```
def build(self):
    # ... 前面的程式碼 ...
    self.current_selection = []

    for i in range(16):
        card = Button(
            text='?',
            font_size=24,
            background_color=(1, 0.8, 0.8, 1)
        )
        card.bind(on_press=self.on_card_press)
        card.card_value = self.card_values[i]
        card.revealed = False
        self.cards.append(card)
        self.game_grid.add_widget(card)

    return self.main_layout

def on_card_press(self, instance):
    if instance.revealed:
        return

    instance.text = str(instance.card_value)
    instance.revealed = True
    instance.background_color = (0.8, 0.8, 1, 1)
```

# Practice: 翻牌配對遊戲

- 實作配對檢查邏輯

```
from kivy.clock import Clock

 def on_card_press(self, instance):
        if instance.revealed:
            return

        if len(self.current_selection) < 2:
            instance.text = str(instance.card_value)
            instance.revealed = True
            instance.background_color = (0.8, 0.8, 1, 1)
            self.current_selection.append(instance)

            if len(self.current_selection) == 2:
                Clock.schedule_once(self.check_cards, 1)

    def check_cards(self, dt):
        card1, card2 = self.current_selection
        if card1.card_value == card2.card_value:
            card1.background_color = (0.8, 1, 0.8, 1)
            card2.background_color = (0.8, 1, 0.8, 1)
        else:
            card1.text = '?'
            card2.text = '?'
            card1.revealed = False
            card2.revealed = False
            card1.background_color = (1, 0.8, 0.8, 1)
            card2.background_color = (1, 0.8, 0.8, 1)

        self.current_selection = []
```



# Practice: 翻牌配對遊戲

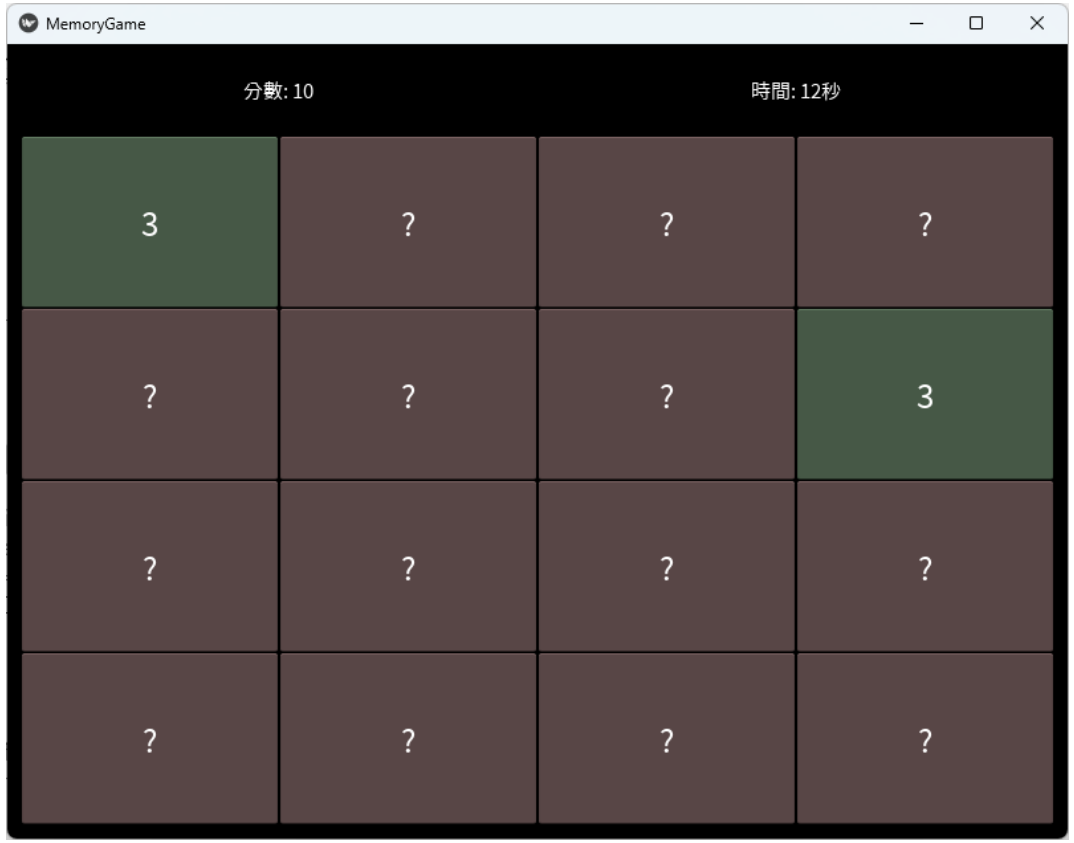
- 加入計分系統

```
def build(self):
    # ... 前面的程式碼 ...

    info_layout = BoxLayout(size_hint_y=None, height=50)
    self.score = 0
    self.score_label = Label(text='分數: 0')
    …

def check_cards(self, dt):
    card1, card2 = self.current_selection
    if card1.card_value == card2.card_value:
        self.score += 10
        self.score_label.text = f'分數: {self.score}'
        # ... 其他配對成功的程式碼 ...
```

# Practice: 翻牌配對遊戲

- 實作計時功能

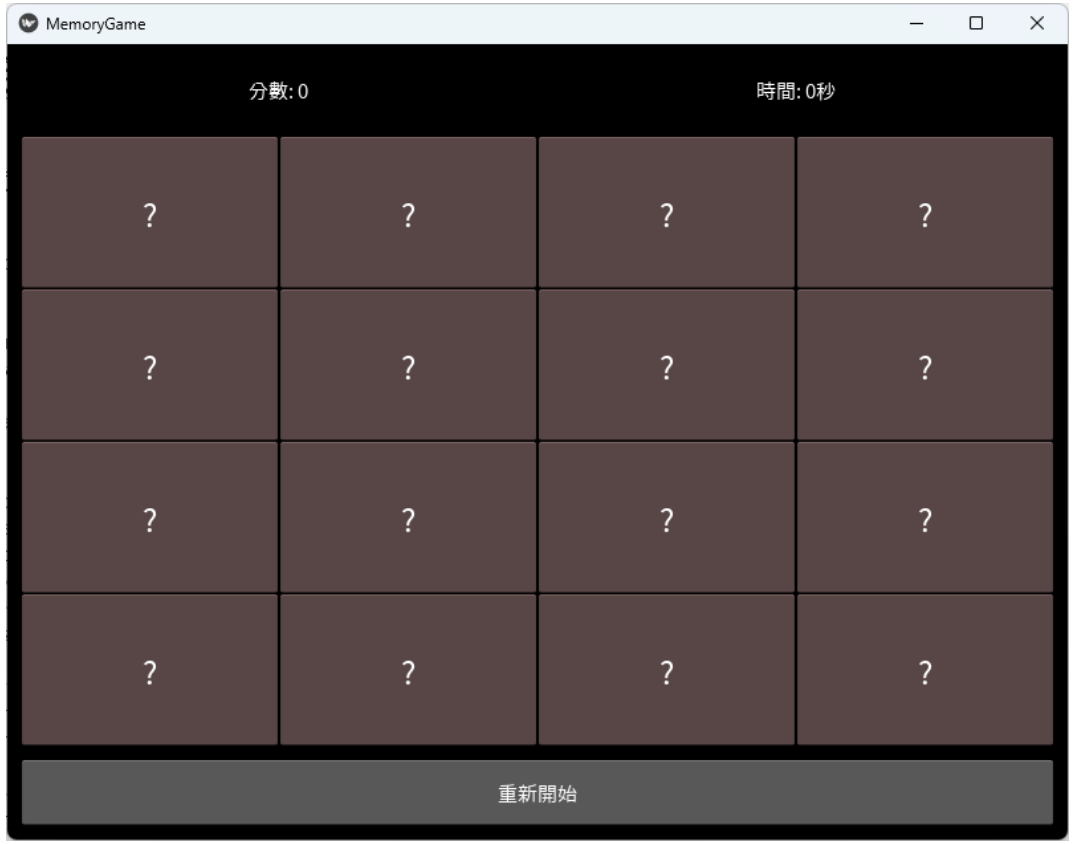
```
def build(self):
    # ... 其他初始化程式碼 ...
    self.time = 0
    self.game_started = False
    self.timer_event = None

def on_card_press(self, instance):
    if not self.game_started:
        self.game_started = True
        self.timer_event = Clock.schedule_interval(self.update_timer, 1)
    # ... 其他點擊邏輯 ...

def update_timer(self, dt):
    self.time += 1
    self.time_label.text = f'時間: {self.time}秒'
```



# Practice: 翻牌配對遊戲

- 加入重新開始功能

```
def build(self):
    # ... 其他布局程式碼 ...
    restart_button = Button(
        text='重新開始',
        size_hint_y=None,
        height=50
    )
    restart_button.bind(on_press=self.restart_game)
    self.main_layout.add_widget(restart_button)

def restart_game(self, size=4):
    self.score = 0
    self.time = 0
    self.game_started = False
    self.game_finished = False
    self.current_selection = []
    self.score_label.text = '分數: 0'
    self.time_label.text = '時間: 0秒'
    self.difficulty = size

    if self.timer_event:
        self.timer_event.cancel()
        self.timer_event = None

    random.shuffle(self.card_values)
    for i, card in enumerate(self.cards):
        card.text = '?'
        card.revealed = False
        card.matched = False  # 重置配對狀態
        card.background_color = (1, 0.8, 0.8, 1)  # 重置為初始顏色
        card.card_value = self.card_values[i]
```



# HW: 翻牌配對遊戲難度修改

- 請將練習修改成可以選擇重新開始的難度
  - 三種難度: 4x4, 6x6, 8x8