**Kivyچیست؟**

در میان زبان‌های برنامه‌نویسی، پایتون از بهترین‌ها محسوب می‌شود که از دلایل مهم این برتری، پشتیبانی عالی کتابخانه‌های آن از ساخت برنامه‌هایی با عملکرد بهتر، رابط کاربری با قابلیت‌های شگفت‌انگیز، پایداری و امکانات فراوانی است که آنها را از سایر برنامه‌ها متمایز می‌کند.

**یکی از فریم ورک های پایتون، Kivyاست** که به عنوان کتابخانه‌ای اوپن سورس برای توسعه رابط کاربر گرافیکی (GUI) به کار می‌رود **و امکان پشتیبانی از چندین پلتفرم** را فراهم می‌کند. بنابراین، در سیستم عامل‌هایی همچونiOS ، ویندوز، اندروید، لینوکس، macOS، »رزبری پای (Raspberry Pi) «و غیره اجرا می‌شوند.

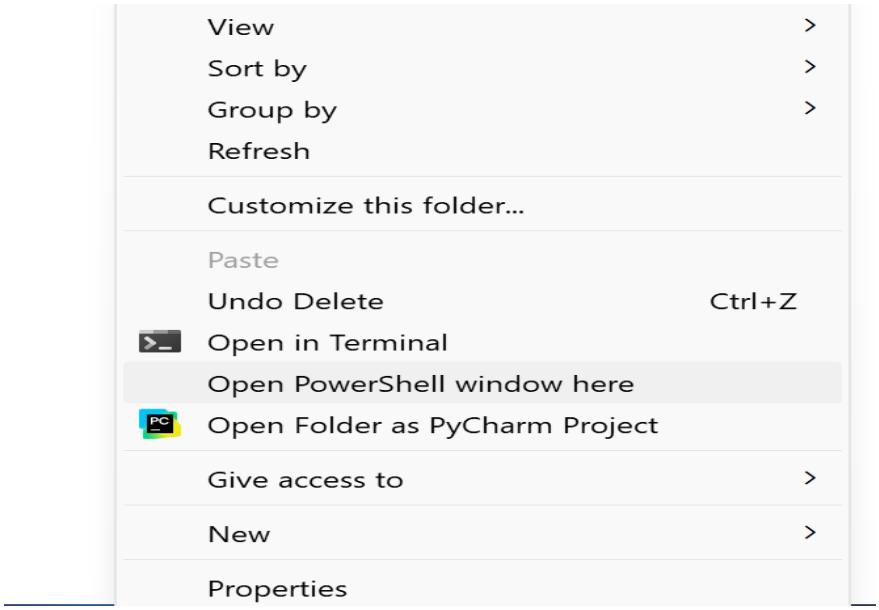
**مزیت اصلی کتابخانهKivy ، مستقل بودن آن از سیستم‌عامل است. این بدان معناست که برنامه‌های توسعه‌یافته با Kivy می‌توانند بر روی انواع مختلفی از پلتفرم‌ها اجرا شوند .**با استفاده از این امکان، توسعه‌دهندگان می‌توانند از کدهای قبلی، بدون نیاز به تغییرات زیاد دوباره استفاده و در پلتفرم‌های مختلف اجرا کنند که این ویژگی باعث سهولت و کارایی در توسعه و اجرای برنامه‌های کاربردی خواهد بود.

**روش نصب Kivy چیست؟**

**Command prompt > Python** اگر پایتون روی سیستم نصب باشه که ورژن پایتون نمایش داده میشه در غیر اینصورت صفحه Microsoft باز شده و اخرین ورژن برای نصب را میتوانیم انتخاب کنید بعد از نصب پایتون باید ( visual studio code) **vscode** را نصب کنید

سپس در فولدری که میخواهید کد های کیوی را داشته باشید **shift کلیک راست + shift** و گزینه **open powershell** را انتخاب کنید .و سپس

**code .** را تایپ کنید تا وارد vscode شوید




در محیط vscode در **تب terminal** تایپ کنید **pip3 install kivy** ( به اینترنت متصل باشید)

**کلاس‌های App و** Label را وارد کنیم. به‌طور کلی، کلاس App عملکرد پایه‌ای مورد نیاز برای ایجاد برنامه‌های رابط کاربری مانند، مدیریت حلقه رویداد را فراهم می‌کند و کلاس Label به عنوان عنصر اصلی بصری یا «Widget» ویجت) برای رابط کاربری ما عمل می‌کند.

**from kivy.app import App**

**from kivy.uix.label import Label**

در ادامه کلاسی با نام MainApp یا هر نام دلخواه دیگری به عنوان «زیرکلاس **(SubClass)** «کلاس App می‌سازیم. در واقع اینجا از مفهوم وارثت استفاده می‌شود و همانطور که می‌دانید در برنامه‌نویسی شی‌گرا، «زیرکلاس (SubClass) «می‌تواند ویژگی‌ها و رفتارهایی که در «کلاس اصلی **(SuperClass) »** تعریف شده‌اند را به ارث ببرند.

به‌طور کلی، **برای ایجاد رابط کاربری در برنامه نیاز است که مِتُد build را** تعریف کنیم که خروجی خود را به عنوان یک شیء از نوع «ویجت (Widget) «یا «چیدمان (layout) «باز می‌گرداند که به عنوان شیء اصلی در ساختار رابط کاربری شما عمل می‌کند. به عبارت دیگر این متد به منظور نمایش دادن هر چیزی می‌باشد که به‌طور مثال در این نمونه، خروجی متد، لیبلی با نوشته «Hello World» است.

```
from kivy.app import App

from kivy.uix.label import Label
```

```
class MainApp(App):
    def build(self):
        return Label(text="Hello, World!")

MainApp().run()
```

**ویجت‌ها**

»ویجت **(Widget) «عنصر اصلی تشکیل دهنده رابط کاربر گرافیکی است که برای نمایش اطلاعات و اجزای مختلف رابط کاربری استفاده می‌شوند** و به کاربران این اجازه را می‌دهد که با رابط کاربر گرافیکی برنامه تعامل داشته باشند. برخی از پرکاربردترین ویجت‌های رابط کاربر گرافیکی در برنامه‌های مبتنی بر Kivi شامل موارد زیر هستند.

**»برچسب :(Label) «**این ویجت به منظور نمایش متن در صفحه نمایش به کار می‌رود.

» **ورودی متنی** :**(Text Input) «**این ویجت به منظور نمایش متنی قابل ویرایش توسط کاربر و با هدف دریافت اطلاعات متنی از او استفاده می‌شود که ویژگی‌های زیر را پیشتیبانی می‌کند.

» **دکمه :(Button) «**این نوع ویجت، یکی از عناصر رابط کاربری را فراهم می‌کند که کاربر با فشردن آن، عملیات یا دستوری مشخص را اجرا می‌کند.

»**چک‌باکس** :**(CheckBox) «**این ویژگی یکی از المان‌های رابط کاربری است که امکان انتخاب یا عدم انتخاب یک وضعیت را برای کاربر فراهم می‌کند.

» **تصویر :(Image) «**ویجت تصویر برای نمایش تصویر در رابط کاربر گرافیکی شما استفاده می‌شود.

»**نوار پیشرفت** :(**ProgressBar)«**این المان از رابط کاربر گرافیکی، برای به نمایش گذاشتن پیشرفت کارها استفاده می‌شود و فقط حالت افقی را پشتیبانی می‌کند. این المان حالت تعاملی ندارد و فقط یک ویجت نمایشگر است.

**«کشویی» (** :(**DropDown**این ویجت به صورت یک فهرست کشویی نمایش داده می‌شود و به کابر این امکان را می‌دهد یکی از موارد فهرست را انتخاب کند.

## Styled Label

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

kivy.require('1.11.1')

class B(App):
    def build(self):
        return Label(text='Styled Label', font_size='24sp', color=(0.2, 0.6,
0.9, 1))

B().run()
```

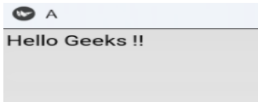
## Multiline and Alignment

```
        return Label(
            text='Line one\nLine two\nLine three',
            font_size='18sp',
            halign='center',
            valign='middle',
            text_size=(300, None)
        )
```

## Markup Text

```
        return Label(
            text='[b]Bold[/b] and [color=ff3333]Red[/color] text',
            markup=True,
            font_size='20sp'
        )
```


```
from kivy.app import App
from kivy.uix.textinput import TextInput

class A(App):
    def build(self):
        return TextInput(text='Type here', multiline=False)

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

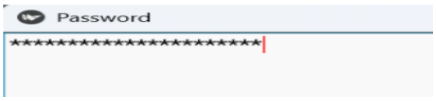



# multiple lines

```
        return TextInput(hint_text='Type multiple lines here', multiline=True)
```


# password

```
        return TextInput(hint_text='Enter password', password=True,
multiline=False)
```




**نمایش تصویر**

```
from kivy.app import App

from kivy.uix.image import Image

class MainApp(App):

    def build(self):

        img = Image(source='/path/to/Faradars.png',

                    size_hint=(1, .5),

                    pos_hint={'center_x':.5, 'center_y':.5})


        return img

MainApp.run()
```

## «چیدمان(Layout)»

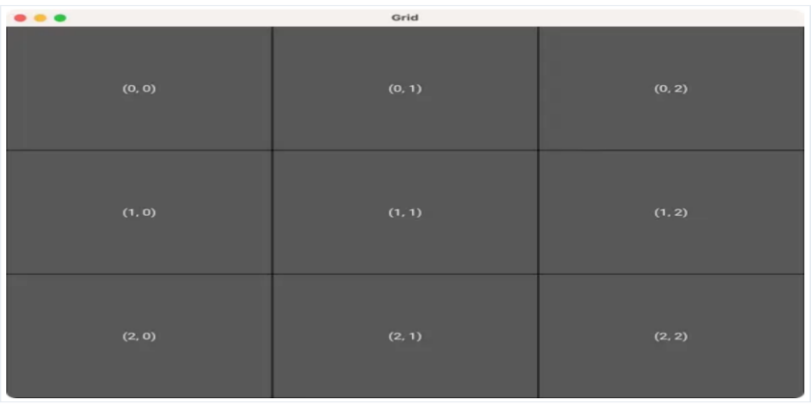
ریمورک Kivy مجموعه‌ای از کلاس‌های چیدمان را نیز فراهم می‌کند که به شما امکان می‌دهد که ویجت‌ها را به صورت کاربردی و منسجم ترتیب دهید .

یکی از کلاس‌های معمول «چیدمان (Layout) «با نام GridLayout

با استفاده از این کلاس می‌توان شبکه‌ای از ردیف‌ها و ستون‌ها ایجاد کرد. به بیانی دیگر این چیدمان، فضای موجود را گرفته و آن را به ستون‌ها و سطرها تقسیم می‌کند، سپس ویجت‌ها را به سلول‌های حاصل شده اضافه می‌کند

# GridLayout

```
from kivy.app import App

from kivy.uix.button import Button

from kivy.uix.gridlayout import GridLayout

ROWS = COLS = 3

class GridApp(App):

    def build(self):

        root = GridLayout(rows=ROWS, cols=COLS)

        for i in range(ROWS):

            for j in range(COLS):

                root.add_widget(Button(text=f"({i}, {j})"))

        return root

GridApp().run()
```



# BoxLayout

```
import random

from kivy.app import App

from kivy.uix.button import Button

from kivy.uix.boxlayout import BoxLayout

red = [1,0,0,1]

green = [0,1,0,1]

blue =  [0,0,1,1]

purple = [1,0,1,1]

class HBoxLayoutExample(App):

    def build(self):

        layout = BoxLayout(padding=10)

        colors = [red, green, blue, purple]

        for i in range(5):

            btn = Button(text="Button #%s" % (i+1),
background_color=random.choice(colors)                    )

            layout.add_widget(btn)

        return layout

HBoxLayoutExample .run()
```

## افزودن Events ها

مانند اکثر ابزارهای رابط کاربری، فریمورک Kivy نیز مبتنی بر «رویداد» ( Events) است. این بدان معناست که برنامه‌های کاربردی Kivy **به رویدادهای کاربر از جمله، کلید فشرده شده، رویداد ماس و لمس پاسخ می‌دهند.**

```
from kivy.app import App
from kivy.uix.button import Button
class MainApp(App):
    def build(self):
        button = Button(text='Hello from Kivy',
                        size_hint=(.5, .5),
                        pos_hint={'center_x': .5, 'center_y': .5})
        button.bind(on_press=self.on_press_button)

        return button

    def on_press_button(self, instance):
        print('You pressed the button!')
MainApp.run()
```

**با استفاده از متد `bind()` ، رویداد on_press به متد `on_press_button` ارتباط داده شده است. به بیان دیگر با فشرده شدن دکمه توسط کاربر، متد `on_press_button` فراخوانی می‌شود که برای مثال در این کد، نوشته مورد نظر چاپ می‌شود.**

-

## موقعیت هر دکمه را با فشردن هر دکمه روی همون دکمه نشان دهد

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

```
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.image import Image
from kivy.core.window import Window
from kivy.uix.gridlayout import GridLayout

r = 3
c = 3
x = 50

class m(App):
    def build(self):
        root = GridLayout(rows=r, cols=c)
        for i in range(r):
            for j in range(c):
                root.add_widget(Button(text=f"({i}, {j})"))
        return root

m().run()
```

# طراحی باتن و تصویر و label

```
from kivy.app import App
from kivy.core.window import Window
from kivy.uix.widget import Widget
from kivy.uix.button import Button
```

```
from kivy.uix.label import Label
from kivy.uix.image import Image


class M(App):
    def build(self):
        root = Widget()
        cx = Window.width / 2
        cy = Window.height - 120
        LB = Label(text='hello', color=(1, 1, 1, 1), pos=(cx + 100, cy - 180))
        img = Image(source=r"C:\Users\asus\Desktop\my.png", size_hint=(None,
None), size=(120, 120), pos=(20, Window.height - 180))
        b = Button(text='press', size=(120, 40), pos=(cx, cy))


        def ok(instance):
            LB.text = 'you press'


        b.bind(on_press=ok)
        root.add_widget(img)
        root.add_widget(LB)
        root.add_widget(b)
        return root


M().run()
```

طراحی شطرنج با باتن

```
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
```

```
r = 6
c = 6

class m(App):
def build(self):
root = GridLayout(rows=r, cols=c)

for i in range(r):
for j in range(c):

if (i + j) % 2 == 0:
color = (1, 1, 1, 1)
else:
color = (0, 0, 0, 1)

btn = Button(
background_normal="",
background_color=color
)

root.add_widget(btn)

return root
m().run()
```

**طراحی کیبورد**

```
def build (self):
    root = GridLayout(rows=2)
```

```
    layout = GridLayout(rows=2,cols=3)
    textbt = textinput (text ='')
    root.add_widget(textbt)
    Labels = ['a','b','c','d','e','f']      تعریف ارایه ه
    for i in Labels
        btn = Button(text=i)
        btn = bind(on_press = ok)
        layout.add_widget(btn)
    root.add_widget(layout)
    return root
def ok (self,instance)
    self.textbt.text + = instance.text
myapp().run()
```

**بافشردن هر دکمه رنگ دکمه به رنگ متن آن دکمه تغییر کند**

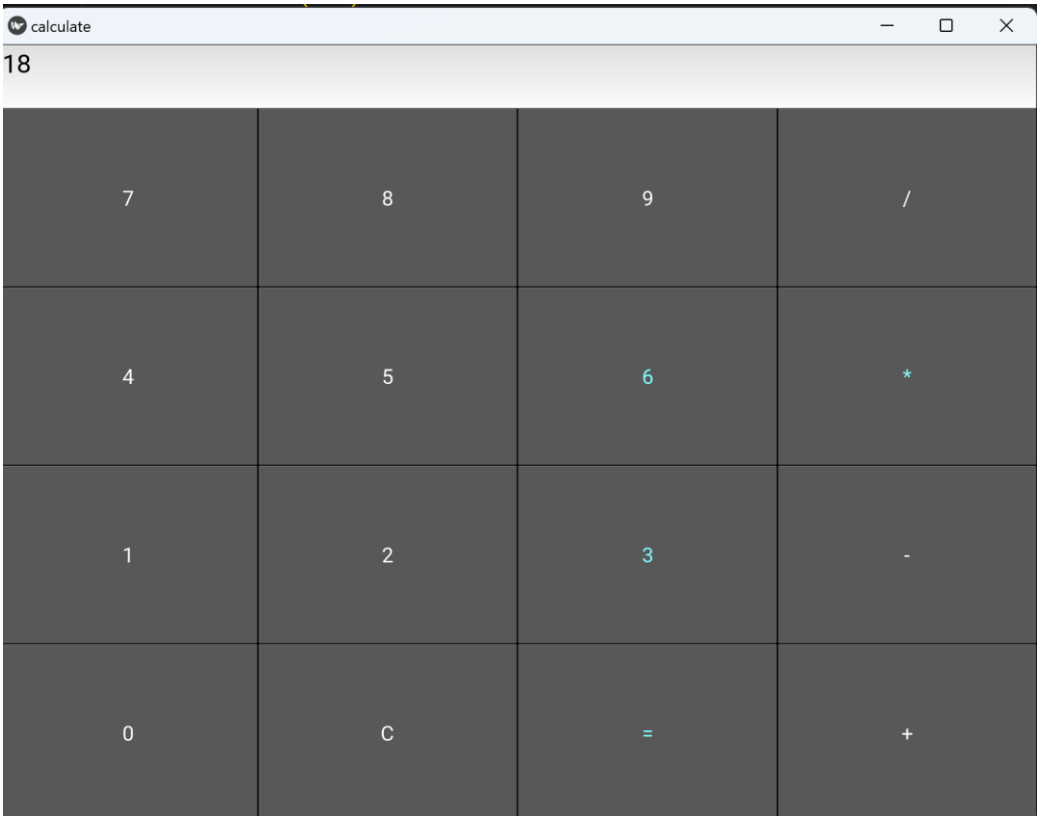
```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
```

```
class m(App):
    def build(self):
        root = GridLayout(rows=2, cols=2)

        labels = ["blue", "red", "green", "black"]

        colors = {
            "blue":  (0, 0, 1, 1),
            "red":   (1, 0, 0, 1),
            "green": (0, 1, 0, 1),
            "black": (0, 0, 0, 1)
        }

        def add(instance):
            instance.background_color = colors[instance.text]

        for name in labels:
            btn = Button(text=name)
            btn.bind(on_press=add)
            root.add_widget(btn)

        return root

m().run()
```

## برنامه طراحی ماشین حساب

```
from kivy.app import App

from kivy.uix.gridlayout import GridLayout

from kivy.uix.button import Button

from kivy.uix.textinput import TextInput


class calculate(App):

    def build(self):

        root = GridLayout(rows=2, cols=1)



        display = TextInput(text='', multiline=False, font_size=40, size_hint_y=None,
height=100)

        root.add_widget(display)



        buttons = [

            '7','8','9','/',

            '4','5','6','*',

            '1','2','3','-',
```

```
            '0','C','=','+'
        ]

        grid = GridLayout(cols=4)
        for t in buttons:
            btns = Button(text=t, font_size=32)
            def ok(instance, b=btns):
                if b.text == 'C':
                    display.text = ''
                elif b.text == '=':
                    display.text = str(eval(display.text))

                else:
                    display.text = display.text + b.text

                b.color = [.51, 1, 1, 1]

            btns.bind(on_press=ok)
            grid.add_widget(btns)

        root.add_widget(grid)
        return root

calculate().run()
```




برای دسترسی به ویجت مربوطه، آرگومان ورودی Instance داده می‌شود.

-

«تابع داخلی (built-in ) «پایتون به نام eval() برای اجرای محاسبات استفاده می‌کند. برای مثال اگر کاربر فرمول ۱+۲ را به عنوان ورودی وارد کرده باشد آنگاه تابع eval() نتیجه را محاسبه می‌کند و در نهایت این نتیجه به عنوان مقدار جدید برای ویجت solution ثبت می‌شود .

## Popupچیست؟

Popupیک پنجره کوچک است که روی صفحه باز می‌شود

)برای پیام، خطا، هشدار، تأیید، فرم کوچک، (...

مثل پیغام "آیا مطمئن هستید؟"

```
from kivy.app import App
```

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

class MyApp(App):
    def build(self):
        btn = Button(text="نمایش پنجره")
        btn.bind(on_press=self.show_popup)
        return btn

    def show_popup(self, *args):
        pop = Popup(title="پیام",
                    content=Label(text="سلام! این یک Popup است"),
                    size_hint=(0.6, 0.4))
        pop.open()

MyApp().run()
```

## Option Button) دکمه انتخابی: زن، مرد(

در Kivy برای Option Button می‌توانیم از ToggleButton.

ToggleButton — گروهی

دکمه‌ها را در یک گروه قرار می‌دهیم:

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

```
btn_man = ToggleButton(text="مرد", group="gender")
btn_woman = ToggleButton(text="زن", group="gender")
```

چطور بفهمیم کدام انتخاب شده؟

هر دکمه اگر انتخاب شود، state == "down" می‌شود.

```
def show_gender(self, instance):
    if btn_man.state == "down":
        print("مرد انتخاب شد")
    elif btn_woman.state == "down":
        print("زن انتخاب شد")
btn_man.bind(on_press=self.show_gender)
btn_woman.bind(on_press=self.show_gender)
```

**برای انتقال بین دو صفحه screenmanager**

```
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, Screen, SlideTransition
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout

class MenuPage(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = BoxLayout(orientation="vertical")

        btn_go = Button(text="رفتن به صفحه دوم")
```

```
        btn_go.bind(on_press=lambda x: self.manager.current="page2")


        layout.add_widget(btn_go)
        self.add_widget(layout)


class Page2(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        layout = BoxLayout(orientation="vertical")


        layout.add_widget(Button(text="شما در صفحه دوم هستید"))


        btn_back = Button(text="برگشت به منو")
        btn_back.bind(on_press=lambda x: self.manager.current="menu")


        layout.add_widget(btn_back)
        self.add_widget(layout)


class MyApp(App):
    def build(self):
        sm = ScreenManager(transition=SlideTransition(duration=0.3))


        sm.add_widget(MenuPage(name="menu"))
        sm.add_widget(Page2(name="page2"))
```

```
        return sm


MyApp().run()
```

## انیمیشن انتقال صفحه

در ScreenManager کد زیر را تنظیم کن:

```
sm = ScreenManager(transition=SlideTransition(direction='left', duration=0.3))
```

انواع انیمیشن‌ها:

- SlideTransition
- FadeTransition
- WipeTransition
- FallOutTransition
- RiseInTransition

مثال:

```
transition=FadeTransition(duration=0.4)
```

## برنامه paint

✓دکمه ۱ → حالت ننقاشی

با حرکت ماوس نقطه‌های **رنگی تصادفی** رسم می‌کند.

✓دکمه ۲ → □ حالت پاک‌کن

روی مسیر ماوس **دایره‌هایی به رنگ پس‌زمینه** می‌کشد → یعنی پاک می‌شود.

```
from kivy.app import App

from kivy.uix.widget import Widget
```

```
from kivy.graphics import Color, Ellipse
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.button import Button
from kivy.core.window import Window
import random

# رنگ پس زمینه
Window.clearcolor = (0.1, 0.4, 0.4, 1)

class DrawArea(Widget):
    mode = "draw"  # draw = نقاشی, erase = پاک‌کن

    def on_touch_move(self, touch):
        if not self.collide_point(*touch.pos):
            return

        if self.mode == "draw":
            # رنگ تصادفی
            r, g, b = [random.random() for _ in range(3)]
            with self.canvas:
                Color(r, g, b)
                Ellipse(pos=(touch.x-5, touch.y-5), size=(10, 10))

        elif self.mode == "erase":
```

```
            # همان رنگ پس زمینه → پاک می‌کند
            with self.canvas:
                Color(*Window.clearcolor)
                Ellipse(pos=(touch.x-8, touch.y-8), size=(18, 18))


class MainApp(App):
    def build(self):
        layout = BoxLayout(orientation="vertical")

        # ناحیه رسم
        self.area = DrawArea()

        # دکمه‌ها
        btns = BoxLayout(size_hint_y=0.15)

        btn1 = Button(text="✏ نقاشی")
        btn2 = Button(text="□ پاک‌کن")

        btn1.bind(on_press=lambda x: self.set_mode("draw"))
        btn2.bind(on_press=lambda x: self.set_mode("erase"))

        btns.add_widget(btn1)
        btns.add_widget(btn2)
```

```
        layout.add_widget(btns)
        layout.add_widget(self.area)
        return layout

    def set_mode(self, mode):
        self.area.mode = mode


MainApp().run()
```

بدون withهم می‌توانی، ولی طولانی‌تر می‌شود:

```
c = self.canvas
c.add(Color(1,0,0))
c.add(Ellipse(pos=(100,100), size=(50,50)))
```

اما:

```
with self.canvas:
    Color(...)
    Ellipse(...)
```

## lambdaچیه؟

lambdaیعنی **تابع کوتاه یک‌خطی**.

مثال معمولی:

```
def add(a, b):
    return a + b
```

نسخه کوتاه همان:

```
add = lambda a, b: a + b
```

در Kivy استفاده می‌کنیم چون داخل bindفقط یک کار کوچک می‌کنیم:

```
btn1.bind(on_press=lambda x: self.set_mode("draw"))
```

یعنی:
وقتی دکمه فشار داده شد، تابع set_mode("draw")را اجرا کن.

## چرا کنار بعضی کدها ستاره (*) هست؟

ستاره یعنی **باز کردن یک لیست/تاپل به چند آرگومان جدا**.

مثال:

```
color = (1, 0, 0, 1)
Color(*color)
```

❌ Color((1,0,0,1))اشتباه

## collide_pointچی هست؟

collide_point(x, y)بررسی می‌کند آیا مختصات ماوس **داخل ویجت** قرار دارد یا نه.

مثال:

```
if self.collide_point(touch.x, touch.y):
    print("ماوس داخل این ویجت است")
```

برای این استفاده می‌شود تا وقتی بیرون از منطقه نقاشی کلیک می‌کنی، نقاشی نکشد.

در کد کوتاه‌تر:

```
if not self.collide_point(*touch.pos):
    return
```