

# Kivy:

## Cross-platform App Development for Pythonistas

**Build and distribute beautiful Python cross-platform GUI apps with ease.**



# \$ whoami

Mirko Galimberti  
misl6

Edit profile

10 followers · 2 following

Monza, Italy  
me@mirkogalimberti.com  
https://www.mirkogalimberti.com  
@m1sl6

Achievements

Organizations

399 contributions in the last year

Learn how we count contributions

Overview   Repositories 44   Projects   Packages   Stars

Pinned

- kivy-ios Public Forked from kivy/kivy-ios Toolchain for compiling Python / Kivy / other libraries for iOS Python
- python-for-android Public Forked from kivy/python-for-android Turn your Python application into an Android APK Python ⭐ 4
- py4web Public Forked from web2py/py4web HTML
- kivy Public Forked from kivy/kivy Open source UI framework for Linux, macOS, Android Python ⭐ 1
- pyobjus Public Forked from kivy/pyobjus Access Objective-C from Python Python
- pydal Public Forked from web2py/pydal A pure Python Database Python



# #kivy-team

## Core Developers

**Mathieu Virbel**

He became a programming expert from working in IT for years before starting with Kivy.  
He's French, and founded Melting Rocks.  
On Discord, he's known as @tito.

**Gabriel Pettier**

He is an Information Systems engineer. He's from France, but currently lives in the Netherlands.  
On Discord, he's known as @tshirtman.

**Akshay Arora**

He is a freelance developer. He is from India.  
On Discord, he's known as @quanon.

**Alexander Taylor**

He is a software engineer, with a little time to make fun graphical interfaces. He lives in the UK.  
On Discord, he's known as @inclement.

**Matthew Einhorn**

He is a developer using Kivy with Python to automate scientific research. He lives in the eastern USA.  
On Discord, he's known as @matham.

**Richard Larkin**

Richard is an educational software developer (B.Sc, Hons) from South Africa. He likes being silly, meditating, music and hugging fluffy things.  
On Discord, he's known as @ZenCODE.

**Andre Miras**

Linux geek and open source addict, he works as a software architect and lives in Spain.  
On Discord, he's known as @AndreMiras.

**Mirko Galimberti**

He is a Full Stack Developer and lives in Italy. Kivy helped him to speed up the App development process while keeping high standards.  
On Discord, he's known as @m1sl6.



#the-real-kivy-team-part-1



**475 + 219 + 66 + 103 + 70 ...** (and counting)

code contributors



#the-real-kivy-team-part-2



Sponsors and  
Supporters

**195**

(and counting)



```
$ history 0 | grep kivy
```

I guess we can consider the commit  
**1f2fb6eb41651e4d10cb51c24a24af66be37606d** as the **Kivy** birthday.

Which is dated **2010-11-03**

Almost **13** years ago. Wow!

But everything started from a project on which **tito** was already working on at that time...

**PyMT** (Python Multitouch)

(<https://github.com/tito/pymt>)

Created in 2007

**16** years ago!



```
$ history 0 | grep kivy
```

*“Kivy is made for today and tomorrow. Novel input methods such as Multi-Touch have become increasingly important. We created Kivy from scratch, specifically for this kind of interaction ...”*

Well, that statement has been added a quite long time ago to our docs, and still look **fresh**, even if the most important things may have changed meanwhile, as the framework evolved (**and is evolving**) with the tech.



Ok ...

Kivy looks like a **robust** and **long-maintained** project.

But ... why we should  
choose **Kivy** for **App  
Development?**

And **how** I can **persuade** my  
coworkers (or **myself**) to drop  
**non-pythonic** ways to **develop**  
mobile and desktop **apps**?

# Kivy is cross-platform

Write once.

*Less code to maintain!*

Deploy  
anywhere.

(Almost)

iOS, Android, macOS, Linux, Windows





# Kivy is fast

*Production ready!*

Time-critical functionalities are implemented in **Cython**.

GPU Accellerated (when it makes sense)

Intelligent algorithms to minimize costly operations



# Kivy is business friendly

**Kivy** is released under the **MIT License** and is **100%** free to use and is professionally developed, backed and maintained.

**Companies** and **individual** are using **Kivy** for their projects **everyday**.

Completely **FREE** and  
**NO HIDDEN COSTS!** \*

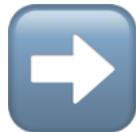
\* Just consider making source code contributions as a thank you to the community!



# Kivy makes Pythonistas Happier

As a Pythonistas, we're open-minded, so we're probably good to switch to another language to develop (mobile or desktop) GUI APPs.

But, if we can just avoid that ...



“An happier software developer is less prone to generate bugs”



A photograph of a man bungee jumping from a red bridge. He is wearing a blue t-shirt, blue jeans, and a blue safety harness. He is suspended in mid-air, arms outstretched, with a red bungee cord attached to his harness. The background is a dense forest of green trees.

TAKE A FULL  
BREATHE ...



\* SAFELY

... AND LET'S  
DISCOVER THE  
KIVY ECOSYSTEM \*

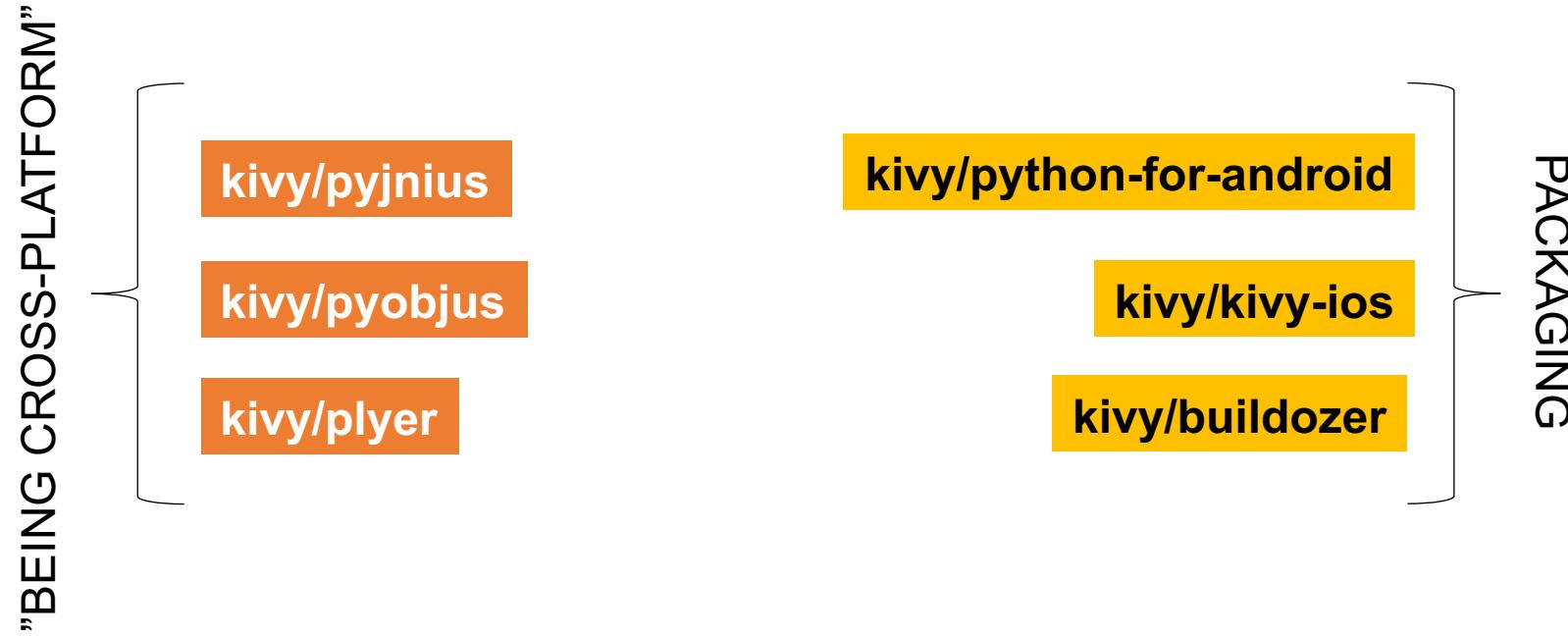
# Kivy has a complete toolset

**“kivy/kivy is just the visible part of the iceberg”**

*Mirko, 2022-08-10 22:22, Rotterdam*



# The toolset: Overview



(An overview of the most important parts of the Kivy ecosystem, you can find and discover other nice projects at <https://www.github.com/kivy>)



# The toolset: kivy/pyjnius

A Python module to access Java classes as Python classes using the Java Native Interface (JNI).

```
from android跑到 runnable import run_on_ui_thread
from jnius import autoclass

@run_on_ui_thread
def show_toast(self):
    # Load the necessary Android (and Java) classes
    Toast = autoclass('android.widget.Toast')
    String = autoclass('java.lang.String')

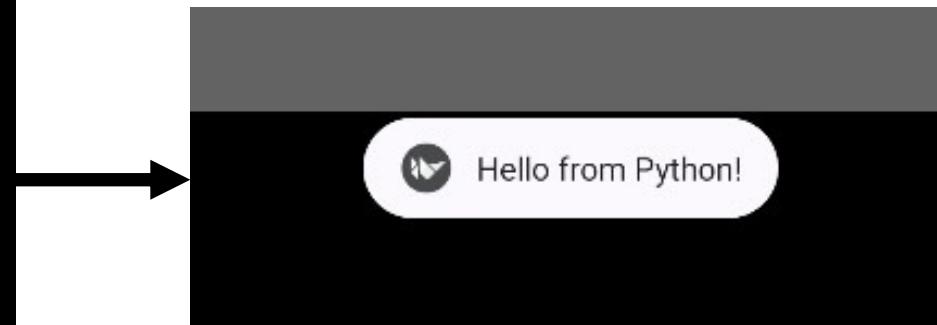
    # Get the current Android context
    current_context = autoclass('org.kivy.android.PythonActivity').mActivity

    # Create a Java string and pass it to the Toast
    msg = String("Hello from Python!")

    # Create a Toast message
    toast = Toast.makeText(current_context, msg, Toast.LENGTH_SHORT)
    # Display the Toast message
    toast.show()
```

Access **Android APIs** and  
third-party **Android libraries** from **Python**!

Access **Java** classes on  
your **desktop** environment!



# The toolset: kivy/pyobjus

Python module for accessing Objective-C classes as Python classes using Objective-C runtime reflection.

Access **native** and third-party **APIs** on **macOS** and **iOS** from **Python** !

```
from pyobjus import autoclass, objc_str
from pyobjus.dylib_manager import load_framework, INCLUDE

# load AppKit framework into pyobjus
load_framework(INCLUDE.AppKit)

# get NSAlert class
NSAlert = autoclass('NSAlert')

# create an NSAlert object, and show it.
alert = NSAlert.alloc().init()
alert.setMessageText_(objc_str('Hello world!'))
alert.runModal()
```



# The toolset: kivy/plyer

Pythonic alternative to  
**access platform-specific**  
features!

Plyer is a platform-independent API to use features commonly found on various platforms, notably mobile ones, in Python.

```
import plyer

# The most basic and exhaustive example (in the world)
plyer.tts.speak("Hello World")
```

Accelerometer - Audio recording – Barometer – Battery – Bluetooth – Brightness – Call – Camera – Compass - CPU count – Devicename – Email – Flash – GPS – TTS, and so on ...  
(full list with OS compatibility on [github.com/kivy/plyer](https://github.com/kivy/plyer))

Leverages `pyjnius` and `pyobjus` when needed.



## The toolset: kivy/python-for-android

A packaging tool for **Python** apps on **Android**.

You can create your own Python distribution with the needed **modules** and **dependencies**, and **bundle** it in an **APK** or **AAB** along with your own **code** and **assets**.

## The toolset: kivy/kivy-ios

A packaging tool for **Python** apps on **iOS**.

You can create your own Python distribution with the needed **modules** and **dependencies**, and bundle it in an **APP** along with your own **code** and **assets**.

**So these tools are just copying my dependencies and code into an artifact that can run on Android or iOS?**

**Unfortunately is not that easy.**

*(for us)*



## Whiteboard session.

Why is not that easy to package a  
Python mobile APP?

( under the hood )



**We need to package a Python interpreter that is able to run on Android or iOS**

**We need to start the Python interpreter**

**Non-plain python packages are not available on PyPi for Android or iOS**

**On iOS, we're required to statically link everything on the main executable.**

**Not everything is available on iOS and Android (like subprocess)**

**Data scientists love libraries based on FORTRAN**

**No, Android doesn't have apt-get**

**No, iOS doesn't have brew**

*And many more ...*

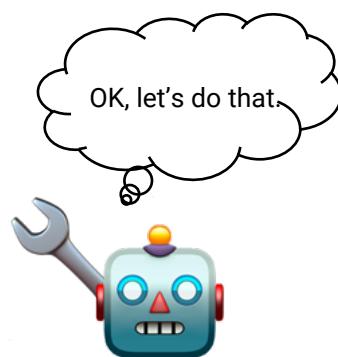
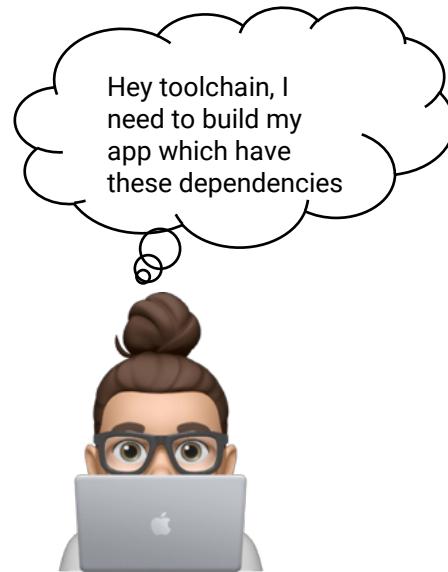


The background of the slide features four clear plastic containers filled with a variety of healthy meal components. Each container includes white rice, green falafel balls, yellow bell pepper cubes, red bell pepper cubes, chickpeas, and leafy greens like kale or collard greens. The containers are arranged in a 2x2 grid.

# How we managed to fix it.

( We can do better, and how the whole Python community can help)

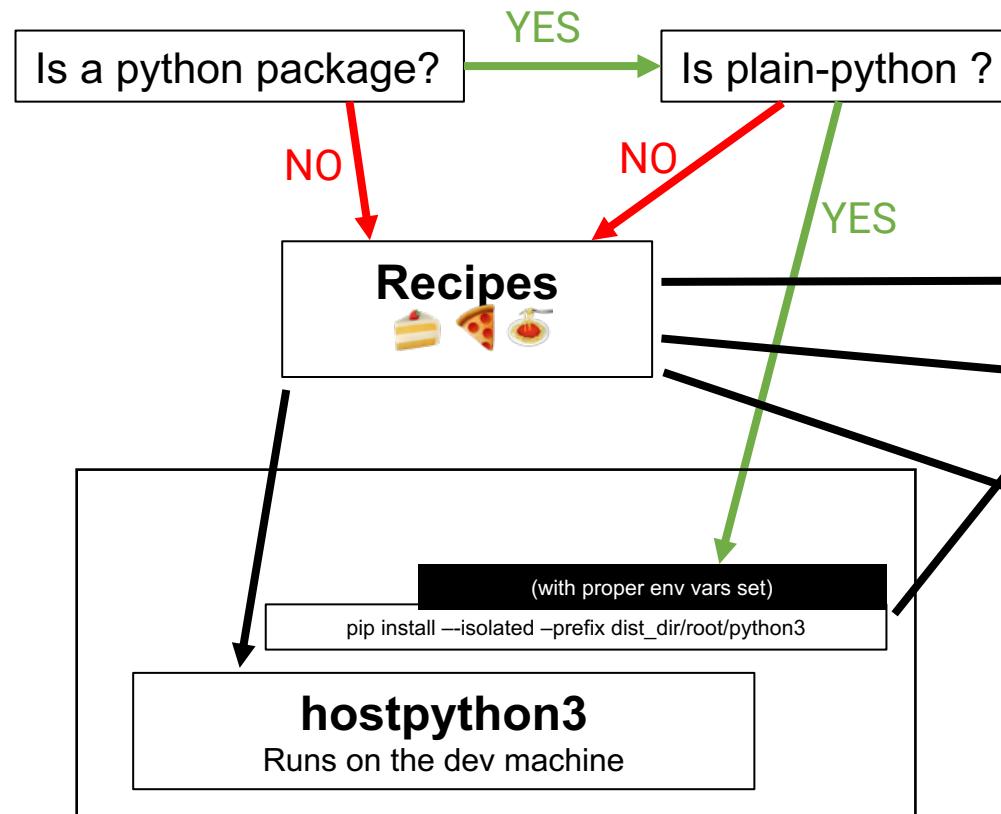
# kivy/kivy-ios and kivy/python-for-android



## How they works

(I'm not talking about installation and usage, that's too easy.)

A dependency graph is built, and then, for every dependency ...



Android or iOS APP

**bootstrap**

(Android: SDL2, webview, service – iOS:  
SDL2)

**python3**

Runs on the target device

**Python Packages**

('site-packages')

**libs**

(openssl, sdl2, ... )

**YourApp**

Your APP code and assets



# The toolset: kivy/buildozer

A tool for creating application packages **easily**.  
(easily == *without worrying about cli flags*)

**One single `buildozer.spec` file in your app directory, describing your application requirements.** buildozer will use that spec to create a package for Android, iOS, macOS.

```
$ buildozer android debug deploy run
```

(Hey buildozer, please build an Android artifact in debug mode, deploy it on the device and run it)

## A small brief about the current status:

---

Android:  Supported  
(Uses kivy/python-for-android under the hood)

macOS:  Partially supported  
(Uses Kivy.app pre-built artifact under the hood)

iOS:  Partially supported  
(Uses kivy/kivy-ios under the hood)

Windows:  Help wanted  
(Alternative: PyInstaller as documented)

Linux:  Help wanted  
(Alternative: PyInstaller)

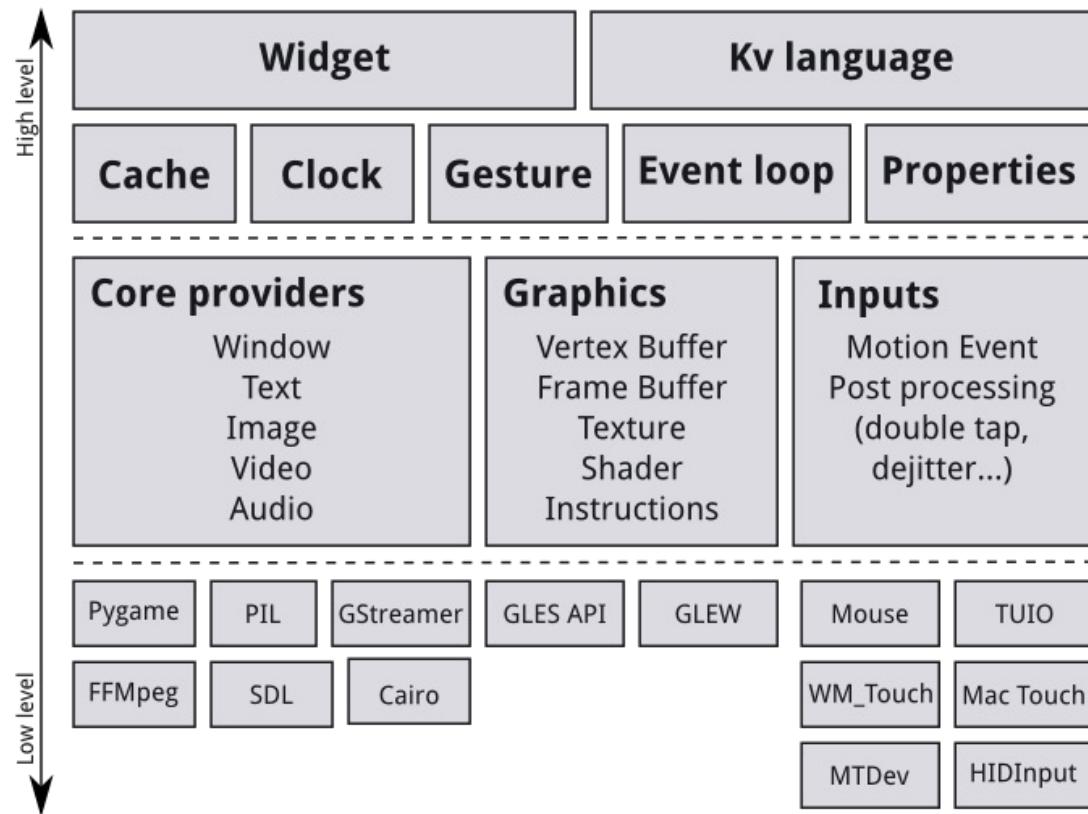


The background of the image is a photograph of a large, white iceberg floating in dark blue ocean water. The sky above is filled with heavy, grey clouds. In the foreground, there is a white rectangular box containing text.

(a brief introduction to)

# The visible part of the iceberg: kivy/kivy

# Kivy architecture



Kivy **abstracts basic tasks** such as:

- Opening a window
  - Displaying **images**
  - Displaying **text**
  - Playing **audio**
  - Getting video feed from a **camera**.
- ... and so on

This makes the **API** both **easy to use** and **easy to extend**.

**Most importantly, it allows us to use specific providers** for the respective scenarios in which your app is being run.

As an example, on macOS, Linux and Windows, there are **different native APIs for the different core tasks**.

On every platform, **Kivy chooses the best core provider** to use, and you do **not need to worry about it**.



# KVLang (Kivy Design Language)

The **KV language** (KVlang), allows the developer to create the widget tree in a **declarative way** and to bind widget properties to each other or to callbacks in a natural manner.

It allows for very **fast prototypes** and agile changes to your UI.

It facilitates **separating the logic** of your application and its **User Interface**.

```
...  
mylay = BoxLayout(size_hint=(1,.5), orientation="vertical")  
  
salutation_lbl = Label(text="")  
  
mylay.add_widget(salutation_lbl)  
  
btn = Button(text="Say Hi",  
on_release=salutation_lbl.text="Hi!")  
  
mylay.add_widget(btn)  
...
```

```
...  
BoxLayout:  
    size_hint: 1, .5  
    orientation: 'vertical'  
    Label:  
        id: salutation_lbl  
        text: ""  
    Button:  
        text: "Say Hi!"  
        on_release: salutation_lbl.text = "Hi!"  
...
```



# Properties

NumericProperty

StringProperty

ListProperty

ObjectProperty

BooleanProperty

BoundedNumericProperty

OptionProperty

AliasProperty

DictProperty

ColorProperty

ReferenceListProperty

VariableListProperty

ConfigParserProperty

These properties implement the **Observer pattern**,  
and to use them, **you have to declare them at  
class level**.

**Kivy Properties** help you to:

- **Easily manipulate widgets** defined in the Kv language
- Automatically **observe** any **changes** and **act accordingly**
- **Check** and **validate** values
- Optimize memory management

```
class MyClass(EventDispatcher):  
    click_count = NumericProperty(0)  
  
    def inc_click_count(self):  
        self.click_count += 1  
  
    def on_click_count(self, *kwargs):  
        print("Clicked!", self.click_count)
```

Each property by default provides  
an on\_<propertynname> event that is called  
whenever the property's state/value changes.



# Clock

```
while True:  
    say_hi()  
    time.sleep(10)
```



```
from kivy.clock import Clock  
  
def say_hi(dt):  
    print(f"Hi! @ {dt}")  
  
say_hi_ev = Clock.schedule_interval(say_hi, 10)  
  
# say_hi_ev.cancel()
```

`Clock.schedule_interval(my_callback, 0.5) # call my_callback every 0.5 seconds`

`Clock.schedule_once(my_callback, 5) # call my_callback in 5 seconds`

`Clock.schedule_once(my_callback) # call my_callback as soon as possible (usually next frame.)`

`event = Clock.create_trigger(my_callback)  
event() # will run the callback once before the next frame`

Useful to **schedule** a function **call** in the future, once or repeatedly at specified intervals.



# Widget: Events

A **widget** has 2 default types of events:

- **Widget-defined event:** e.g. an event is fired when the Button is released or pressed. (via `dispatch`)
- **Property event:** if your widget changes its position or size, an event is fired. (As seen in Kivy Properties)

```
class CustomButton(Button):  
    current_status = StringProperty("unknown")  
  
    def on_release(self, *args):  
        self.current_status = "released"  
  
    def on_press(self, *args):  
        self.current_status = "pressed"  
  
    def on_current_status(self, *args):  
        print("current_status: ", self.current_status)
```



# Widget: Canvas

Each widget has a **canvas**, aka “A place to draw on”.  
The **canvas** is a group of drawing instructions that  
should be **executed whenever there is a change** to  
the widget’s graphical representation.

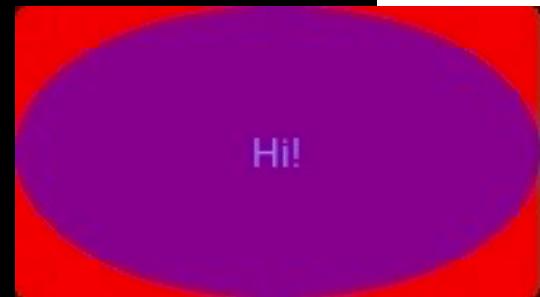
`canvas.before` or `canvas.after` groups can  
be used to **separate instructions** based on **when you  
want them to be executed**.

You can **add instructions** either from **Python** code or  
from the **kv** file.

**TIP:** If you add them via the kv file, the advantage is  
that they are **automatically updated** when any Kivy  
property they depend on **changes**. In Python, you need  
to do this yourself.

```
...
BoxLayout:
    canvas.before:
        Color:
            rgba: 1, 0, 0, 1
        RoundedRectangle:
            pos: self.pos
            size: self.size
            radius: dp(5),
    canvas.after:
        Color:
            rgba: 0, 0, 1, .5
        Ellipse:
            size: self.size
            pos: self.pos
    Label:
        text: "Hi!"
        color: 1,1,1,1
...

```



# built-in UI components

AnchorLayout

BoxLayout

FloatLayout

RelativeLayout

GridLayout

PageLayout

ScatterLayout

StackLayout

Accordion

ActionBar

Bubble

Button

Camera

Carousel

Checkbox

CodeInput

ColorPicker

DropDown

EffectWidget

FileChooser

GestureSurface

Image

Label

ModalView

Popup

ProgressBar

ProgressBar

RecycleView

RstDocument

Scatter

ScreenManager

ScrollView

Slider

Spinner

Splitter

StencilView

Switch

TabbedPanel

TextInput

ToggleButton

TreeView

Video

VideoPlayer

VKeyboard

Widget



# “Roadmap” for a smooth cross-platform experience

(how to avoid painful mistakes)

## Think.

Which **dependencies** I need?

I **really need** all these deps?

Can I use **plain-python alternatives** to non-plain-python dependencies?

I could take an **advantage** by using a **platform-specific implementation**?

## Test.

Test **every dependency**, on **every platform** you want to support (now and in future)

Test **platform-specific implementations**

If the test phase gone **super-smooth**, why are you still here?

## Choose.

A **recipe** to build a non-plain python dependency is **not available**?

Are you **comfortable to patch** your dependency and **write** a recipe? \*

**Anything** you're uncomfortable with and **you want to change**?

\* Consider proposing your recipe and patches upstream

**IMPORTANT:** If you're uncomfortable with a choice you did, re-do the decision process.



# Kivy's community-maintained garden



# Is full of flowers

<https://github.com/kivy-garden>

An organization for developers of Kivy widgets, add-ons and related software.

Some examples:

- **zbarcam**: Real time **Barcode and QR Code scanner** using the camera. It's built on top of Kivy and works with both **pyzbar** or **zbarlight**.
- **mapview**: A **Kivy widget for displaying interactive maps**. It has been designed with lot of inspirations of Libchamplain and Leaflet.
- **graph**: The Graph widget is a **widget for displaying plots**. It supports drawing multiple plot with different colors on the Graph. It also supports a title, ticks, labeled ticks, grids and a log or linear representation on both the x and y axis, independently.

**FREE DOM**

**Kivy provides the bricks.**

Our community a  
garden full of flowers.

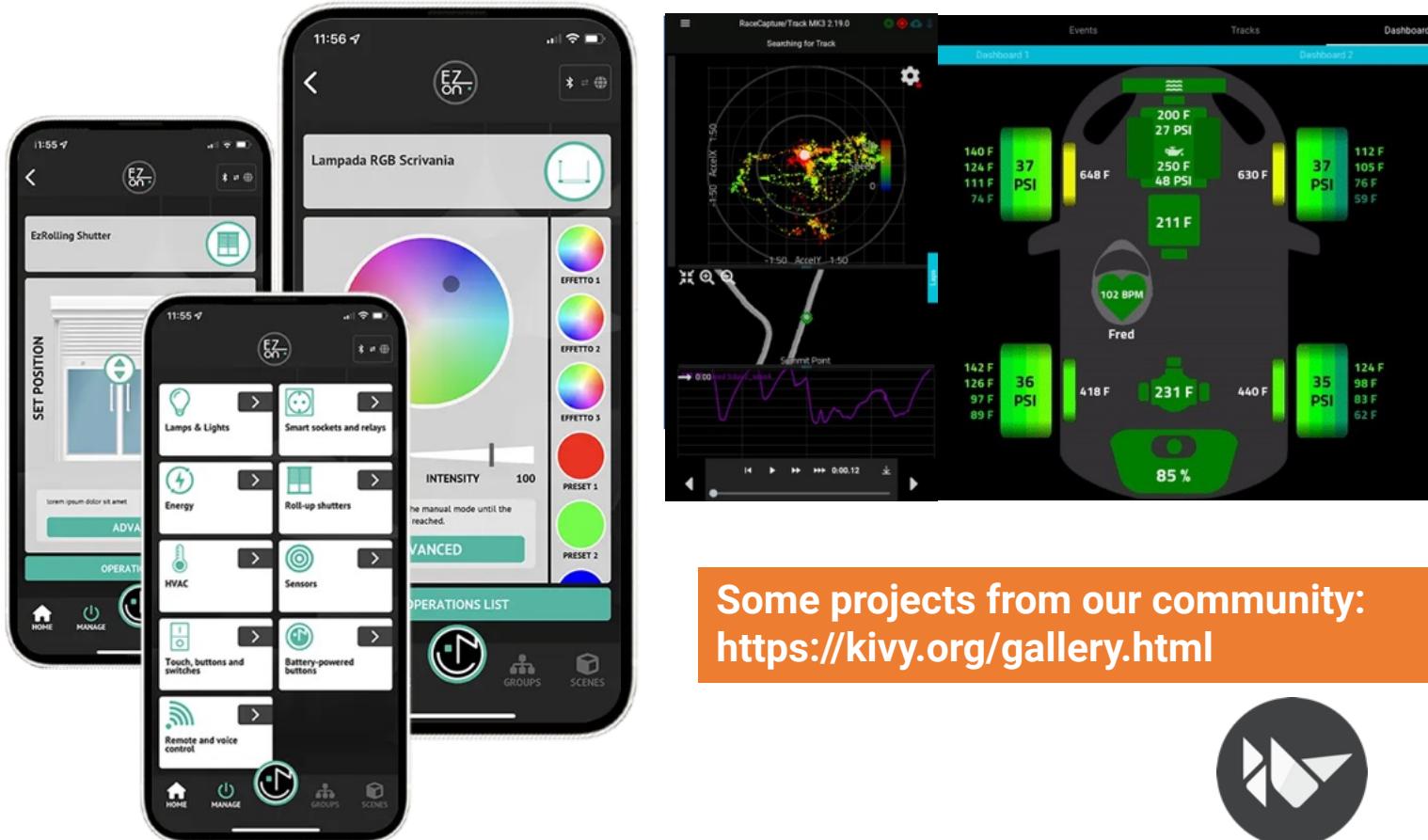
But you can even do more.





Even if Kivy comes with its own set of icons and its UI theme, there's no need to stick on it.

You can **create new widgets and customize existing ones.** (*spoiler: Is easier than writing some CSS3.*)

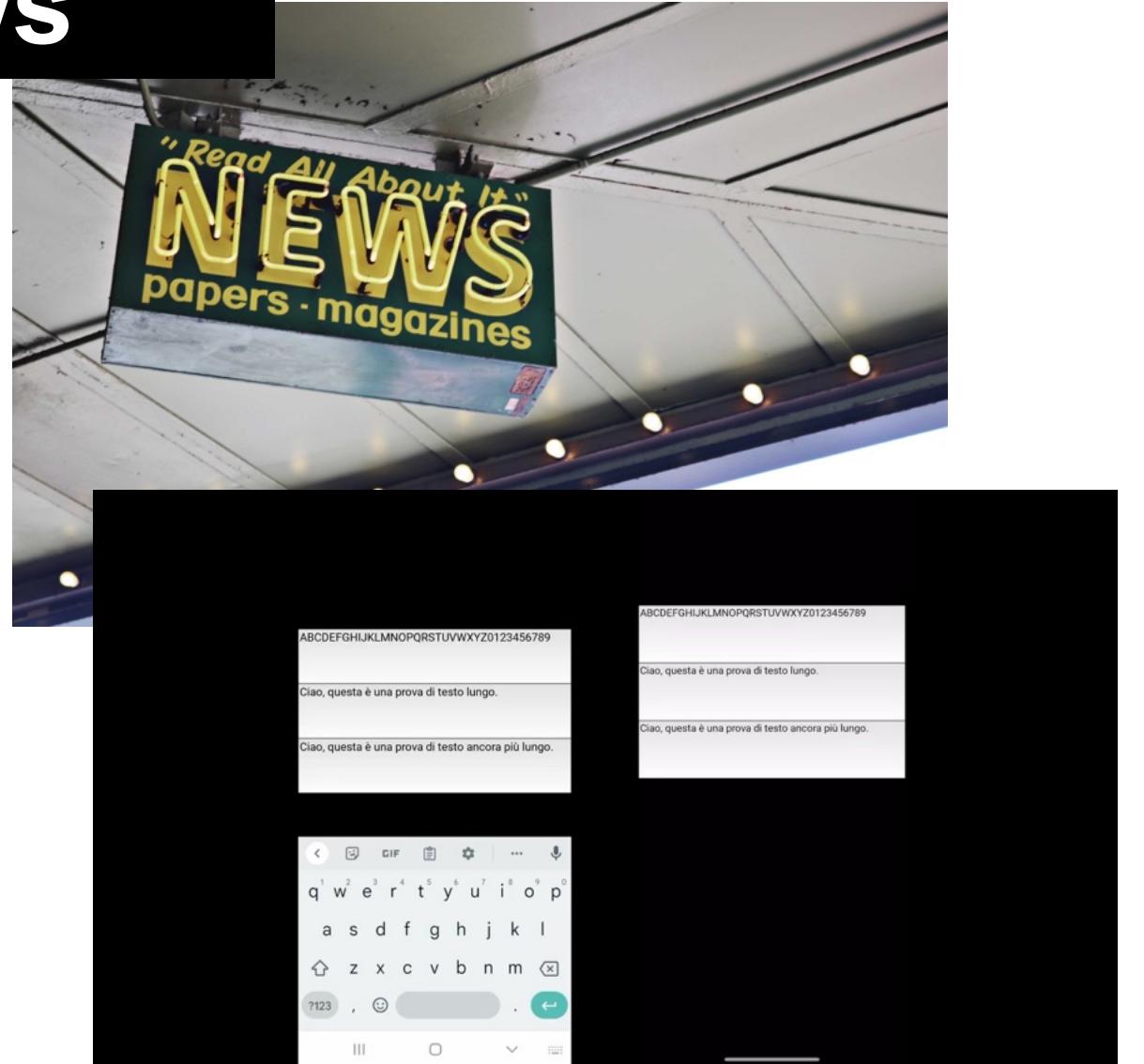


Some projects from our community:  
<https://kivy.org/gallery.html>



# Some fresh news

- 🎉 **Kivy 2.2.1** has been released on 2023-06-17 🎉
- **Kivy-ios v2023.08.24** has been released and supports **Kivy 2.2.1**
- **python-for-android v2023.09.16** has been released and supports **Kivy 2.2.1**
- **Kivy 2.3.0** status 🏗:
  - Some PRs have already been merged!
  - The **brand new TextInput core provider** (currently a POC) is becoming a reality.



# The future.

## Or, at least, some thoughts.

- **Improve the documentation** and guides 📖.
- **Involve** (more) **the community** through meetups and livestreams.
- **Improve the support** for non-latin languages (In 2.2.0 we now use harfbuzz to handle reshaping, but there's still a lot of work to do)
- **Update the camera** 📸 implementation on both **Android** and **iOS**.
- **Together is better:** Involve the whole Python community **making it aware of mobile platforms**.



Python Swiss Summit (LIVE) special:

Need help to get your package ready for mobile platforms? Ping me IRL!





[www.kivy.org](http://www.kivy.org)



[chat.kivy.org](http://chat.kivy.org)



@kivyframework

# Thank You!

Want to talk about Kivy?  
Feel free to ping me IRL to start chatting 

