KIVY

The Open Source
Python App Development Framework

tutorialspoint

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Kivy is a Python library that helps you to build cross-platform GUI applications for Windows, Linux, iOS as well as Android. Kivy supports touch-enabled input. All the widgets in Kivy GUI framework have the capability to handle multi-touch gestures.

In this tutorial, you will learn to build attractive GUI applications with Kivy GUI library. This tutorial is a good starting point for anybody who wants to develop cross-platform GUI apps.

Audience

This tutorial is a good starting point for anybody who wants to develop cross-platform GUI apps. Kivy is also a good choice for mobile app development for Android an iOS devices.

One can follow the examples in this tutorial to develop attractive desktop GUI applications and mobile apps.

Prerequisites

Since kivy is a Python library, an intermediate level knowledge of Python programming is required. Good understanding of principles of Object-oriented programming will be an added advantage.

For Android-based apps, you need to have a basic understanding of how the APKs are built and deployed.

Disclaimer & Copyright

Table of Contents

Kivy Basic Concepts

1. Kivy – Getting Started

Kivy is an open-source Python library. It allows you to build multi-touch applications with a natural user interface (NUI). With Kivy, you can develop cross-platform applications. The same code, written once, can be deployed on different various operating system platforms such as Windows, macOS, Linux, Android, and iOS.

Popular GUI Frameworks in Python

Kivy is one of the many GUI frameworks available in the Python ecosystem. Some of the popular Python libraries for building desktop GUI applications are:

- **Tkinter**: Tkinter package is bundled with Python's standard library. It is the standard Python interface to the Tcl/Tk GUI toolkit.

- **PyQt5**: This library is a Python port for the Qt GUI toolkit. Our extensive tutorial on PyQt5 can be accessed here.

- **WxPython**: WxPython library allows Python programmer to have access to WxWidgets, an open-source GUI toolkit, originally written in C++. To learn more about WxPython, click here.

- **Kivy**: Kivy is a Python library that helps you to build cross-platform GUI applications for Windows, Linux, iOS as well as Android. Kivy supports touch-enabled input. All the widgets in Kivy GUI framework have the capability to handle multi-touch gestures.

Kivy is equipped with powerful graphics and multimedia features. A Kivy app can support audio, video, animations, 2D as well as 3D graphics.

Key Features of Python Kivy

Here are some key features of Python Kivy:

- Kivy supports touch-enabled input. All the widgets in Kivy GUI framework have capability to handle multi-touch gestures.

- Kivy's comprehensive GUI widgets and robust layout management makes designing attractive interfaces easily possible.

- Kivy is equipped with powerful graphics and multimedia features. This makes it possible to incorporate, 2D as well as 3D graphics, animations, audio and video componenets in the application.

- Various types of input devices are supported by Kivy. It includes touch, mouse and gestures.

- Kivy API can access mobile device hardware components such as camera, GPS, etc.

- Kivy uses OpenGL ES 2 graphics library, and is based on Vertex Buffer Object and shaders.

- Kivy relies upon Cython for its core implementation, and SDL2 (Simple DirectMedia Layer) for low-level multimedia and input handling.

To deploy the Kivy application on desktops with Windows, Linux or iOS operating systems, the distributable is built with PyInstaller. To build an APK for Android, you need to use Android Studio and Buildozer utility.

The Kivy Language

Kivy uses a special declarative language called Kivy Language (sometimes also called Kv language) to build user interface layouts for Kivy applications. It serves the purpose of separating the design aspect of an app from its programming logic. The design is written in a text file with ".kv" extension. Kivy framework automatically loads the ".kv" file and builds the UI based on the specifications given in it.

The initial version of Kivy library was released in 2011. Currently, Kivy version 2.2 is available, which has been released in May 2023.

2. Kivy – Installation

To build a Kivy application, you need to have Python installed in your computer. Kivy 2.2.0, the latest stable version, officially supports Python versions 3.7 to 3.11. If Python is not already installed, download the installer of latest Python version, appropriate for your operating system and architecture, from Python's official website - https://www.python.org/downloads/

Python Virtual Environment

Python recommends the use of virtual environment to avoid conflicts with other Python versions and packages.

A virtual environment allows us to create an isolated working copy of Python for a specific project without affecting the outside setup. We shall use the "venv" module in Python's standard library to create virtual environment. PIP is included by default in Python version 3.4 or later.

Creating a Virtual Environment

Use the following command to create a virtual environment in Windows:

```
C:\users\user\>python -m venv c:\kivyenv
```

On Ubuntu Linux, update the APT repo and install "venv", if required, before creating a virtual environment.

```
mvl@GNVBGL3:~ $ sudo apt update && sudo apt upgrade -y
mvl@GNVBGL3:~ $ sudo apt install python3-venv
```

Then, use the following command to create a virtual environment:

```
mvl@GNVBGL3:~ $ sudo python3 -m venv kivyenv
```

Activating a Virtual Environment

You need to activate the virtual environment. On Windows, use the following command:

```
C:\>cd kivyenv
C:\kivyenv>scripts\activate
(kivyenv) C:\kivyenv>
```

On Ubuntu Linux, use the following command to activate the virtual environment:

```
mvl@GNVBGL3:~$ cd kivyenv
mvl@GNVBGL3:~/kivyenv$ source bin/activate
(myenv) mvl@GNVBGL3:~/kivyenv$
```

Installing Kivy Using the "pip" Utility

The easiest way to install any Python package is with the use of "pip" utility. Python 3 installation comes with the "pip" installer. After activating the virtual environment, use the following command from the CMD terminal in Windows or Linux terminal:

```
pip3 install "kivy[base]" kivy_examples
```

This installs the Kivy package with minimal dependencies. The "kivy_examples" package is optional. Instead of "base", the "full" option enables audio/video support.

Installing the Dependency Libraries for Kivy

SDL2 (Simple DirectMedia Layer) is a major dependency for Kivy. On Windows OS, SDL2 is automatically installed when you use the "pip" utility. However, for Linux and macOS, you need to install SDL2 separately.

On macOS, you can install SDL2 using Homebrew by running the following command in your terminal:

```
brew install sdl2
```

If on Linux OS, use the corresponding package manager to install SDL2. For example, it is done with the following command on Ubuntu Linux machine:

```
sudo apt-get install libsdl2-dev
```

Additionally, you may have to install other dependencies such as "gstreamer" and "Pillow" for certain specific features of Kivy.

Verifying the Kivy Installation

To verify if Kivy has been properly installed, start the Python interactive shell and import the package. The console shows that the Kivy dependencies are also imported.

```
>>> import kivy
[INFO] [Logger] Record log in
C:\Users\mlath\.kivy\logs\kivy_23-05-26_0.txt
[INFO] [deps] Successfully imported "kivy_deps.gstreamer" 0.3.3
[INFO] [deps] Successfully imported "kivy_deps.angle" 0.3.3
[INFO] [deps] Successfully imported "kivy_deps.glew" 0.3.1
[INFO] [deps] Successfully imported "kivy_deps.sdl2" 0.6.0
[INFO] [Kivy] v2.2.0
[INFO] [Kivy] Installed at "c:\kivyenv\Lib\site-
packages\kivy\__init__.py"
[INFO] [Python] v3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023,
16:38:35) [MSC v.1934 64 bit (AMD64)]
[INFO] [Python] Interpreter at "c:\kivyenv\Scripts\python.exe"
[INFO] [Logger] Purge log fired. Processing...
[INFO] [Logger] Purge finished!
```

You can also obtain the list of all the packages installed using the "pip freeze" command:

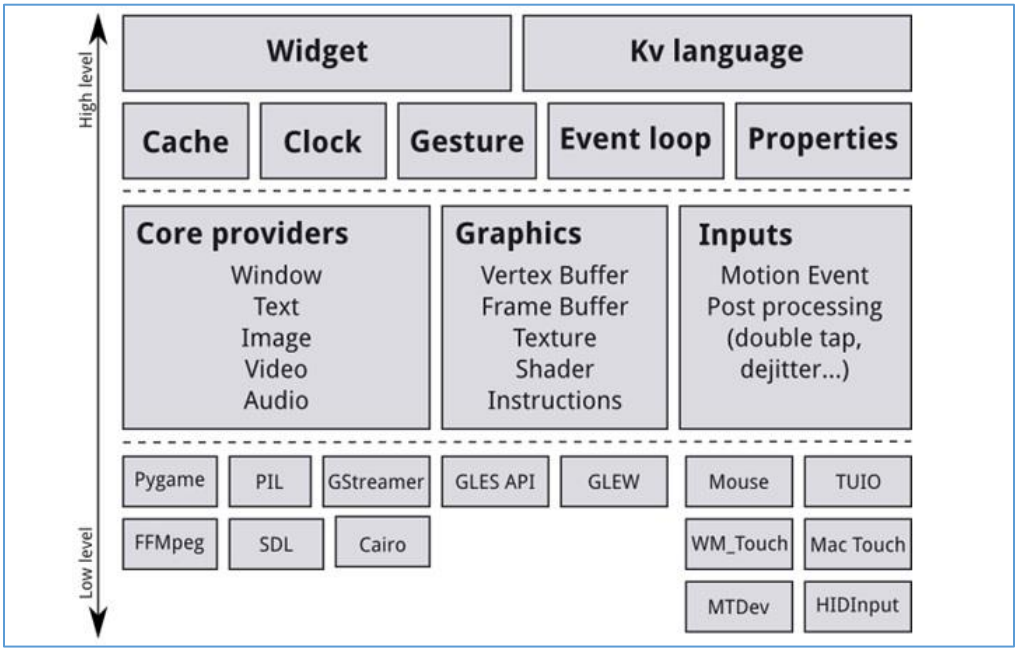
```
(kivyenv) C:\kivyenv>pip3 freeze
certifi==2023.5.7
charset-normalizer==3.1.0
docutils==0.20.1
idna==3.4
Kivy==2.2.0
kivy-deps.angle==0.3.3
kivy-deps.glew==0.3.1
kivy-deps.gstreamer==0.3.3
kivy-deps.sdl2==0.6.0
Kivy-examples==2.2.0
Kivy-Garden==0.1.5
Pillow==9.5.0
```

```
Pygments==2.15.1
pypiwin32==223
pywin32==306
requests==2.31.0
urllib3==2.0.2
```

3. Kivy – Architecture

Read this chapter to understand the design architecture of Kivy framework. On one end, Kivy provides various widgets that lets the user to interact with the application, and on the other end, it interacts with the various hardware devices such as mouse TUIO, audio and video streams, etc. The middle layer consists of drivers or providers for processing touch inputs, audio and video, graphics instructions and text input.

This is the official architectural diagram of the Kivy framework:



Core Providers

An important feature of Kivy architecture is "modularity" and "abstraction". The actions such as opening a window, reading audio and video stream, loading images, etc., are the core tasks in any graphical application. Kivy abstracts these core tasks by providing easy to implement API to the drivers that control the hardware.

Kivy uses specific providers for operating system on which your app is being run. Each operating system (Windows, Linux, MacOS, etc.) has its own native APIs for the different core tasks. The act as an intermediate communication layer between the operating system on one side and to Kivy on the other. Thus Kivy fully leverages the functionality exposed by the operating system to enhance the efficiency.

Use of platform-specific libraries reduces the size of the Kivy distribution and makes packaging easier. This also makes it easier to port Kivy to other platforms. The Android port benefited greatly from this.

Input Providers

An input provider is a piece of code that adds support for a specific input device. The different input devices having built-in support in Kivy include:

- Android Joystick Input Provider
- Apple's trackpads
- TUIO (Tangible User Interface Objects)
- mouse emulator
- HIDInput

To add support for a new input device, provide a new class that reads your input data from your device and transforms them into Kivy basic events.

Graphics

OpenGL is base of the entire Graphics API of Kivy framework. OpenGL instructions are used by Kivy to issue hardware-accelerated drawing commands. Kivy does away the difficult part of writing OpenGL commands by defining simple-to-use functionality.

Kivy uses OpenGL version 2.0 ES (GLES or OpenGL for embedded systems) with which you can undertake cross-platform development.

Core Library

The high level abstraction is provided by the following constituents of Kivy framework:

- **Clock**: the Clock API helps you to schedule timer events. Both one-shot timers and periodic timers are supported.

- **Gesture Detection**: An important requirement of multitouch interfaces. The gesture recognizer detects various kinds of strokes, such as circles or rectangles. You can even train it to detect your own strokes.

- **Kivy Language**: The kivy language is used to easily and efficiently describe user interfaces. This results in separation of the app design from developing the application logic.

- **Properties**: Kivy's unique concept of property classes (they are different from property as in a Python class) are the ones that link your widget code with the user interface description.

UIX

Kivy's user interface is built with widgets and layouts.

- Widgets are the UI elements that you add to your app to provide some kind of functionality. Examples of widget include buttons, sliders, lists and so on. Widgets receive MotionEvents.

- More than one widgets are arranged in suitable layouts. Kivy provides layout classes that satisfy the requirement of placement of widgets for every purpose. Examples would be Grid Layouts or Box Layouts. You can also nest layouts.

Event Dispatch

The term "widget" is used for UI elements in almost all graphics toolkits. Any object that receives input events is a widget. One or more widgets are arranged in a tree structure.

The Kivy app window can hold only one root widget, but the root widget can include other widgets in a tree structure. As a result, there is a "parent-children-sibling" relationship amongst the widgets.

Whenever a new input event occurs, the root widget of the widget tree first receives the event. Depending on the state of the touch, the event is propagated down the widget tree.

Each widget in the tree can either process the event or pass it to the next widget in hierarchy. If a widget absorbs and process the event, it should return True so that its propagation down the tree is stopped and no further processing will happen with that event.

```
def on_touch_down(self, touch):
    for child in self.children[:]:
        if child.dispatch('on_touch_down', touch):
            return True
```

Since the event propagates through the widget tree, it is often necessary to verify if the event has occurred in the area of a certain widget that is expected to handle it. The collide_point() method can help in ascertaining this fact. This method checks if the touch position falls within the 'watched area' of a certain widget and returns True or False otherwise. By default, this checks the rectangular region on the screen that's described by the widget's **pos** (for position; x & y) and **size** (width & height).

4. Kivy – File Syntax

Kivy framework provides a concise and declarative approach to define the widget structure and appearance, with the use of Kivy Language (also known as **Kv language**). It is a declarative language, used exclusively for building user interfaces in Kivy applications. Its main advantage is that you can separate the UI design from the application logic written in Python.

The UI design is defined in a text file which must have a ".kv" extension. It contains the hierarchical sequence of widgets in the application window. The file adapts a tree-like structure, showing the parent-child-sibling relationship among the widgets. Below each widget, its properties, events and event handlers are specified.

The **kv** design language stipulates the following conventions while creating a ".kv" file, so that Python and the Kivy framework can identify and load the appropriate widget structure:

- Name of the file must be in lowercase
- It must match with the main class in your application. This class is inherited from the App class.
- If the name of the class ends with "app" or "App" (for example, **HelloApp**), the ".kv" file must exclude "app" from its name. It means, for HelloApp class, the name of the ".kv" file must be "hello.kv".
- The ".kv" file must be in the same folder in which the Python application file (.py) is present.

While using the ".kv" file, the App class doesn't override the build() method. Declaring a class simply with a pass statement is enough. When the run() method is invoked, Kivy automatically loads the UI from the respective ".kv" file.

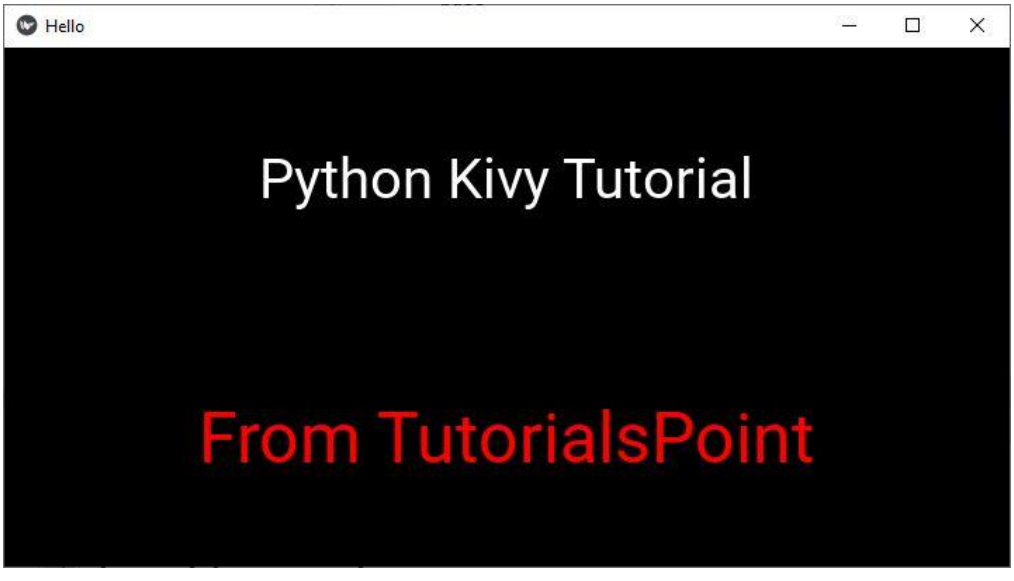
Let us first remove the build() method from the HelloApp class:

```
from kivy.app import App


class HelloApp(App):
    pass


app = HelloApp()
app.run()
```

The User interface is defined in "hello.kv" file in the same folder. We have a top level BoxLayout with vertical orientation, under which two labels are placed. Save the following script as "hello.kv" file

```
BoxLayout:
    orientation: 'vertical'
    Label:
        text: 'Python Kivy Tutorial'
        font_size: '30pt'
    Label:
        text: 'From TutorialsPoint'
        font_size: '50'
        color: (1,0,0,1)
```

Now, if you run the "hello.py" program, it will produce the following output:



In latter chapters, we shall learn how to add event handlers to the widgets in the ".kv" file.

==========

End of ebook preview

If you liked what you saw…

Buy it from our store @ https://store.tutorialspoint.com