# ***Что такое Kivy?***

Kivy - библиотека Python с открытым исходным кодом для быстрой разработки приложений, использующих инновационные пользовательские интерфейсы, такие как мульти-тач приложения.

Kivy работает на Linux, Windows, OS X, Android, iOS, and Raspberry Pi. Вы можете запустить один и тот же код на всех поддерживаемых платформах.

Оно может изначально использовать большинство входов, протоколов и устройств, включая WM\_Touch, WM\_Pen, Mac OS X Trackpad и Magic Mouse, Mtdev, Linux Kernel HID, TUIO. В комплект входит симулятор мультитач-мыши.

Kivy на 100% свободен в использовании, под лицензией MIT (начиная с 1.7.2) и LGPL 3 для предыдущих версий. Инструментарий профессионально разработан, поддерживается и используется. Вы можете использовать его в коммерческом продукте.

Фреймворк стабилен и имеет хорошо документированный API, а также руководство по программированию, которое поможет вам начать работу. Графический движок построен поверх OpenGL ES 2, используя современный и быстрый графический конвейер.

В набор инструментов входит более 20 виджетов, все с высокой степенью расширения. Многие части написаны на C с использованием Cython и протестированы с помощью регрессионных тестов.

Используя Kivy на своем компьютере, вы можете создавать приложения, которые работают на:

* Настольные компьютеры: OS X, Linux, Windows.
* устройства iOS: iPad, iPhone.
* Android устройства: планшеты, телефоны.
* Любые другие сенсорные профессиональные/доморощенные устройства, поддерживающие TUIO (Материальные объекты пользовательского интерфейса).

Самый простой способ установить Kivy – это с помощью pip.

Приложение Kivy всегда начинается с инициализации и запуска класса App. Именно он отвечает за создание окон, интрефейса с OS, а также предоставляет входную точку при создании элементов интерфейса пользователя.

from kivy.app import App

class TicTacToeApp(App):

pass

if \_\_name\_\_ == "\_\_main\_\_":

TicTacToeApp().run()

Уже это приложение можно запустить, вы увидите пустое черное окно.

Виджеты — это отображаемые на экране элементы управления, которыми пользователь может оперировать. Любой инструментарий графического интерфейса пользователя поставляется с набором виджетов. Типичными представителями виджетов, что вы не раз использовали, являются кнопки, выпадающие списки и вкладки. Внутри фреймворка Kivy встроено много виджетов.

Каждый из виджетов Kivy представляет собой простой графический элемент со своим функционалом от стандартного - кнопки, лейблы, текстовые поля ввода, или вспомогательные элементы для позиционирования дочерних, до более абстрактных - диалоги выбора файлов, камера или видео проигрыватель. Большинство основных виджетов Kivy легко сочетаются друг с другом.

Так как "Hello, world!" уже стало практически неизбежным примером, давайте поскорее его рассмотрим:

from kivy.uix.label import Label

class TicTacToeApp(App):

def build(self):

return Label(text="Hello World!",

font\_size=100,

color=(0, 1, 0, 1))

Запуск программы «Hello, Kivy!»

Принцип работы Kivy можно уловить, взглянув на следующее приложение «Hello, World!»:

Python

from kivy.app import App

from kivy.uix.label import Label

class MainApp(App):

def build(self):

label = Label(text='Hello from Kivy',

size\_hint=(.5, .5),

pos\_hint={'center\_x': .5, 'center\_y': .5})

return label

if \_\_name\_\_ == '\_\_main\_\_':

app = MainApp()

app.run()

from kivy.app import App

from kivy.uix.label import Label

class MainApp(App):

def build(self):

label = Label(text='Hello from Kivy',

size\_hint=(.5, .5),

pos\_hint={'center\_x': .5, 'center\_y': .5})

return label

if \_\_name\_\_ == '\_\_main\_\_':

app = MainApp()

app.run()

Каждому приложению Kivy требуется создать подкласс App и переопределить метод build(). Сюда вы помещаете код UI или вызываете другие функции, которые определяют код UI. В данном случае создается виджет Label и передается text, size\_hint и pos\_hint. Последние два аргумента не обязательны.

size\_hint говорит Kivy о размерах что нужно использовать при создании виджета. Используются два числа:

Первое число x указывает на размер ширины элемента управления.

Второе число y указывает на размер высоты элемента управления.

Значение обоих чисел должно быть в промежутке между 0 и 1. Значение по обоих показателей по умолчанию равно 1. Также можно задействовать pos\_hint, что используется для позиционирования виджета. В коде, размещенном выше, указывается, что виджет должен быть размещен в центре осей x и y.

Для запуска приложения нужно инициализировать класс MainApp и вызвать метод run(). После этих действий на экране появится следующее:

черный экран с надписью посередине

Kivy также выводит в stdout довольно много текста:

много текста

Это может быть полезно для отладки приложения.

Далее добавим виджет Image и посмотрим, чем он отличается от Label.

Отображение виджета Image в Kivy Python

В Kivy есть несколько видов виджетов, связанных с изображениями. Для загрузки картинок с жесткого диска можно задействовать Image:

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12  13  14 | from kivy.app import App  from kivy.uix.image import Image    class MainApp(App):      def build(self):          img = Image(source='/path/to/real\_python.png',                      size\_hint=(1, .5),                      pos\_hint={'center\_x':.5, 'center\_y':.5})            return img    if \_\_name\_\_ == '\_\_main\_\_':      app = MainApp()      app.run() |

В данном коде импортируется Image из подпакета kivy.uix.image. Класс Image принимает много разных параметров, однако единственным для нас нужным является source, что указывает Kivy, какое изображение должно быть загружено. Здесь передается полный путь к выбранному изображению.

Разметка (Layout) в UI Kivy

У каждого фреймворка есть свой собственный метод для размещения виджетов.

В Kivy за это отвечают Лейауты (Layouts). Доступно несколько различных типов Лейаутов. Чаще всего используются следующие виды:

* BoxLayout;
* FloatLayout;
* GridLayout.

Калькулятор работает следующим образом:

* **В строках с 8 по 10** создается список operators и несколько полезных значений, last\_was\_operator и last\_button, которые будут использованы чуть позже.
* **В строках с 11 по 15** создается лейаут верхнего уровня main\_layout, к нему также добавляется виджет только для чтения TextInput.
* **В строках с 16 по 21** создается вложенный [список](https://python-scripts.com/lists-tuples-dictionaries#list) из списков, где есть большая часть кнопок для калькулятора.
* **В строке 22** начинается цикл for для кнопок. Для каждого вложенного списка делается следующее:
  1. **В строке 23** создается BoxLayout с горизонтальной ориентацией.
  2. **В строке 24** начинается еще один цикл for для элементов вложенного списка.
  3. **В строках с 25 по 39** создаются кнопки для ряда и связываются обработчиком событий, после чего кнопки добавляются к горизонтальному BoxLayout из строки 23.
  4. **В строке 31** этот лейаут добавляется к main\_layout.
* **В строках с 33 по 37** создается кнопка равно (=) и привязывается к обработчику событий, после чего она добавляется к main\_layout.

Далее создается обработчик событий .on\_button\_press().

Почти все виджеты приложения вызывают .on\_button\_press(). Это работает следующим образом:

* **Строка 41** принимает аргумент instance, в результате чего можно узнать, какой виджет вызвал функцию.
* **Строки между 42 и 43** извлекают и хранят значения solution и text кнопки.
* **Строки c 45 по 47** проверяют, на какую кнопку нажали. Если пользователь нажимает с, тогда очищается solution. В противном случае используется утверждение else.
* **Строка 49** проверяет, было ли у решения предыдущее значение.
* **Строки с 50 по 52** проверяют, была ли последняя нажатая кнопка оператором. Если да, тогда solution обновляться не будет. Это необходимо для предотвращения создания двух операций в одном ряду. К примеру, 1 \* / будет недействительным утверждением.
* **Строки с 53 по 55** проверяют, является ли первый символ оператором. Если да, тогда **solution** обновляться не будет, так как первое значение не может быть значением оператора.
* **Строки с 56 по 58** переходят к условию else. Если никакое из предыдущих значений не найдено, тогда обновляется solution.
* **Строка 59** устанавливает last\_button к метке последней нажатой кнопки.
* **Строка 60** устанавливает last\_was\_operator к значению True или False в зависимости от того, был символ оператором или нет.

Последней частью кода будет .on\_solution():

Здесь берется текущий текст из solution и используется встроенный в Python eval() для исполнения. Если пользователь создал формулу вроде 1+2, тогда eval() запустит код и вернет результат. В конце результат устанавливается как новое значение виджета solution.