

Kivy — Создание мобильных приложений на Python

Принципы работы фреймворка Kivy Python

Kivy был создан в 2011 году. Данный кросс-платформенный фреймворк Python работает на Windows, Mac, Linux и Raspberry Pi. В дополнение к стандартному вводу через клавиатуру и мышь он поддерживает [мультикас](#). Kivy даже поддерживает ускорение GPU своей графики, что во многом является следствием использования OpenGL ES2. У проекта есть лицензия MIT, поэтому библиотеку можно использовать бесплатно и вместе с коммерческим программным обеспечением.

Во время разработки приложения через Kivy создается интуитивно понятный интерфейс ([Natural user Interface](#)), или **NUI**. Его главная идея в том, чтобы пользователь мог легко и быстро приспособиться к программному обеспечению без чтения инструкций.

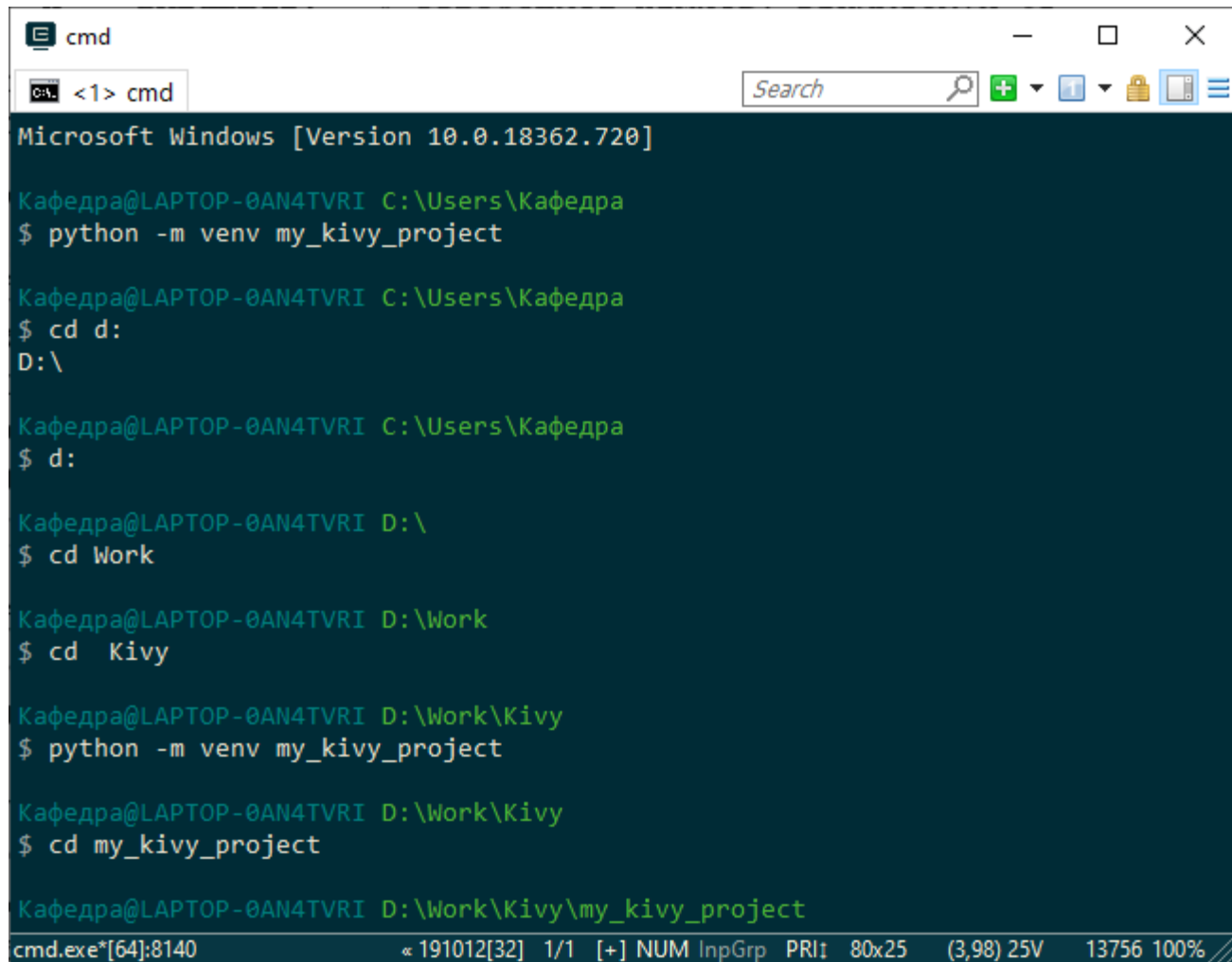
Kivy не задействует нативные элементы управления, или виджеты. Все его виджеты настраиваются. Это значит, что приложения Kivy будут выглядеть одинаково на всех платформах. Тем не менее, это также предполагает, что внешний вид вашего приложения будет отличаться от нативных приложений пользователя. Это может стать как преимуществом, так и недостатком, все зависит от аудитории.

Установка Kivy

У Kivy есть множество зависимостей, поэтому лучше устанавливать его в [виртуальную среду Python](#).

Можно использовать встроенную библиотеку Python [venv](#) или же пакет [virtualenv](#).

Виртуальная среда Python создается следующим образом:



```
cmd
C:\> cmd
Microsoft Windows [Version 10.0.18362.720]
Кафедра@LAPTOP-0AN4TVRI C:\Users\Кафедра
$ python -m venv my_kivy_project

Кафедра@LAPTOP-0AN4TVRI C:\Users\Кафедра
$ cd d:
D:\

Кафедра@LAPTOP-0AN4TVRI C:\Users\Кафедра
$ d:

Кафедра@LAPTOP-0AN4TVRI D:\
$ cd Work

Кафедра@LAPTOP-0AN4TVRI D:\Work
$ cd Kivy

Кафедра@LAPTOP-0AN4TVRI D:\Work\Kivy
$ python -m venv my_kivy_project

Кафедра@LAPTOP-0AN4TVRI D:\Work\Kivy
$ cd my_kivy_project

Кафедра@LAPTOP-0AN4TVRI D:\Work\Kivy\my_kivy_project
cmd.exe*[64]:8140      α 191012[32]  1/1  [+] NUM InpGrp  PRI:  80x25  (3,98) 25V  13756 100%
```

Работа с виджетами в Kivy

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.

Запуск программы «Hello, Kivy!»

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

```
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()
```

Отображение виджета Image в Kivy Python

В Kivy есть несколько видов виджетов, связанных с изображениями. Для загрузки картинок с жесткого диска можно задействовать Image, а при использовании адреса URL подойдет AsyncImage. К следующему примере берется стандартный класс Image:

```
from kivy.app import App
from kivy.uix.image import Image

class MainApp(App):
    def build(self):
        img = Image(source='images/image1.jpg',
                    size_hint=(1, .5),
                    pos_hint={'center_x':.5, 'center_y':.5})

        return img

if __name__ == '__main__':
    app = MainApp()
    app.run()
```

Разметка (Layout) в UI Kivy

У каждого фреймворка есть свой собственный метод для размещения виджетов. К примеру, в wxPython используются классификаторы, а в Tkinter будет задействован лейаут, или менеджер геометрии. В Kivy за это отвечают Лейауты (Layouts). Доступно несколько различных типов Лейаутов. Чаще всего используются следующие виды:

BoxLayout;
FloatLayout;
GridLayout.

Во время создания лейаута следует учитывать следующие аргументы:

padding: Отступ padding между лейаутом и его дочерними элементами уточняется в пикселях. Для этого можно выбрать один из трех способов:

Список из четырех аргументов: [padding_left, padding_top, padding_right, padding_bottom]

Список из двух аргументов: [padding_horizontal, padding_vertical]

Один аргумент: padding=10

spacing: При помощи данного аргумента добавляется расстояние между дочерними виджетами.

orientation: Позволяет изменить значение orientation для BoxLayout по умолчанию — с горизонтального на вертикальное.

```
import kivy
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 #{} {}".format(i+1, random.choice(colors)),
                        background_color=random.choice(colors))

            layout.add_widget(btn)
        return layout

if __name__ == "__main__":
    app = HBoxLayoutExample()
    app.run()
```


Добавление событий в Kivy

Как и многие другие инструментарии GUI, по большей части Kivy полагается на события. Фреймворк отзывается на нажатие клавиш, кнопки мышки или прикосновение к сенсорному экрану. В Kivy задействован концепт Часов (Clock), что дает возможность создать своего рода график для вызова определенных функций в будущем.

В Kivy также используется концепт Свойств (Properties), что работает с EventDispatcher. Свойства помогают осуществить проверку достоверности. Они также запускают события, когда виджет меняет размер или позицию.

```
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('Вы нажали на кнопку!')

if __name__ == '__main__':
    app = MainApp()
    app.run()
```

Использование языка дизайна KV

Kivy предоставляет язык дизайна **KV**, что можно использовать в приложениях Kivy. Язык KV позволяет отделить дизайн интерфейса от логики приложения. Он придерживается принципа [разделения ответственности](#) и является частью архитектурного паттерна [Модель-Представление-Контроллер \(Model-View-Controller\)](#). Предыдущий пример можно обновить, используя язык KV:

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

class ButtonApp(App):
    def build(self):
        return Button()

    def on_press_button(self):
        print('Вы нажали на кнопку!')

if __name__ == '__main__':
    app = ButtonApp()
    app.run()
```

<Button>:

```
text: 'Press me'  
size_hint: (.5, .5)  
pos_hint: {'center_x': .5, 'center_y': .5}  
on_press: app.on_press_button()
```

Вы можете установить все ваши виджеты и лейауты внутри одного или нескольких файлов языка KV. Язык KV также поддерживает **импорт модулей Python** в KV, создавая **динамические классы**, и это далеко не предел. Ознакомиться с полным перечнем его возможностей можно в [гиде Kivy по языку KV](#).

Создание приложения Kivy Python

Создание чего-то полезное несомненно является отличным способом выучить новый навык. Учитывая данное утверждение, давайте используем Kivy при [создании калькулятора](#), который будет поддерживать следующие операции:

- Сложение;
- Вычитание;
- Умножение;
- Деление.

В данном приложении будет использован набор кнопок в своего рода лейауте. В верхней части также будет специальный блок для вывода операций и их результатов. В итоге калькулятор будет выглядеть следующим образом:

```

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.button import Button
from kivy.uix.textinput import TextInput

class MainApp(App):
    def build(self):
        self.operators = ["/", "*", "+", "-"]
        self.last_was_operator = None
        self.last_button = None
        main_layout = BoxLayout(orientation="vertical")
        self.solution = TextInput(
            multiline=False, readonly=True, halign="right", font_size=55
        )
        main_layout.add_widget(self.solution)
        buttons = [
            ["7", "8", "9", "/"],
            ["4", "5", "6", "*"],
            ["1", "2", "3", "-"],
            [".", "0", "C", "+"],
        ]
        for row in buttons:
            h_layout = BoxLayout()
            for label in row:
                button = Button(
                    text=label,
                    pos_hint={"center_x": 0.5, "center_y": 0.5},
                )
                button.bind(on_press=self.on_button_press)
                h_layout.add_widget(button)
            main_layout.add_widget(h_layout)

        equals_button = Button(
            text="=", pos_hint={"center_x": 0.5, "center_y": 0.5}
        )
        equals_button.bind(on_press=self.on_solution)
        main_layout.add_widget(equals_button)

        return main_layout

```

```
def on_button_press(self, instance):
    current = self.solution.text
    button_text = instance.text

    if button_text == "C":
        # Очистка виджета с решением
        self.solution.text = ""
    else:
        if current and (
            self.last_was_operator and button_text in self.operators):
            # Не добавляйте два оператора подряд, рядом друг с другом
            return
        elif current == "" and button_text in self.operators:
            # Первый символ не может быть оператором
            return
        else:
            new_text = current + button_text
            self.solution.text = new_text
    self.last_button = button_text
    self.last_was_operator = self.last_button in self.operators
```

Последней частью кода будет `.on_solution()`:

```
def on_solution(self, instance):  
    text = self.solution.text  
    if text:  
        solution = str(eval(self.solution.text))  
        self.solution.text = 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`.