

Разработка калькулятора на фреймворке Kivy

Агентьев Данила Денисович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной работе рассматривается процесс разработки калькулятора с использованием фреймворка Kivy, предназначенного для создания многоплатформенных приложений. Рассмотрены основные принципы работы с Kivy, его возможности в области графического интерфейса и обработки пользовательских данных. Разработка калькулятора включает как базовые математические функции, так и более сложные операции. Программное обеспечение является примером использования Kivy для создания эффективных и удобных пользовательских приложений.

Ключевые слова: Kivy, разработка приложений, калькулятор, графический интерфейс, многоплатформенность, программирование, Python.

Developing a calculator based on the Kivy framework

Akentev Danila Denisovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This paper examines the process of developing a calculator using the Kivy framework, designed to create multi-platform applications. The basic principles of working with Kivy, its capabilities in the field of graphical interface and user data processing are considered. Calculator development includes both basic mathematical functions and more complex operations. The software is an example of using Kivy to create efficient and user-friendly applications.

Keywords: Kivy, application development, calculator, graphical interface, multiplatform, programming, Python.

1 Введение

1.1 Актуальность

В условиях растущего интереса к разработке приложений с поддержкой различных платформ, использование фреймворков, таких как Kivy, становится всё более актуальным. Kivy позволяет создавать приложения для Windows, Linux, macOS, iOS и Android, что существенно расширяет возможности разработчиков. Разработка калькулятора является хорошей практикой для изучения особенностей работы с фреймворком, а также демонстрации его функционала на простом примере.

1.2 Обзор исследований

Для эффективного освоения языка программирования Python можно изучить различные исследования, посвященные его применению в образовательных целях. Например, Г.С. Сейдаметов рассмотрел особенности использования Python при подготовке будущих инженеров-программистов, акцентируя внимание на методах преподавания и их эффективности [1]. В свою очередь, Э.А. Усеинов проанализировал применение объектно-ориентированного подхода в рамках дисциплины "Язык программирования Python", подчеркивая важность освоения этих концепций для будущих специалистов [2]. Д.В. Шкодырев продемонстрировал возможность интеграции Python и системы компьютерной алгебры SymPy в рамках факультативных занятий по теории чисел, что расширяет область применения Python в математическом образовании [3]. В работе В.В. Буркова и И.В. Бантыша был показан потенциал Python в решении задач математического моделирования для электромеханических систем, что является важной составляющей в подготовке специалистов в области инженерии [4]. Д.В. Лучанинов и А.О. Кизянов в своей работе показали процесс создания игры 2048 на фреймворке Kivy [5].

1.3 Цель исследования

Целью работы является разработка функционального калькулятора с использованием фреймворка Kivy, анализ его возможностей и преимуществ при создании приложений с графическим интерфейсом, а также изучение особенностей кроссплатформенной разработки.

2 Материалы и методы

В работе использованы материалы по программированию на Python, документация Kivy, а также примеры реализации калькуляторов. Методологической основой является практическое внедрение фреймворка в разработку пользовательского интерфейса и обработки данных. Для создания калькулятора использованы стандартные библиотеки Kivy для работы с интерфейсом и пользовательским вводом, а также методы объектно-ориентированного программирования для структурирования кода.

3 Результаты и обсуждение

Начнем разработку с создания файла `main.py` в котором пропишем код.

1. Импорты и библиотека Kivy

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.gridlayout import GridLayout
from kivy.properties import ObjectProperty
from kivy.uix.button import Button
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.core.window import Window
```

Рисунок 1. Импорт

В начале кода производится импорт необходимых классов из библиотеки Kivy. Kivy — это библиотека Python, предназначенная для создания кроссплатформенных приложений с графическим интерфейсом.

- *App*: основной класс для создания приложения Kivy.
- *BoxLayout*, *GridLayout*: контейнеры для размещения виджетов в виде коробок и сетки.
- *ObjectProperty*: служит для привязки данных в Kivy, например, для связи текстового поля ввода с объектом.
- *Button*: Элемент для создания кнопок.
- *ScreenManager*, *Screen*: обеспечивает многократный переход между различными экранами приложения.
- *Window*: управляет настройками окна приложения (например, задает размер).

2. Экран калькулятора

```

class CalculatorScreen(BoxLayout, GridLayout, Screen): 1 usage
    input_value = ObjectProperty(None) # Свойство для доступа к текстовому полю ввода
    history = [] # Список для хранения истории расчетов

    # Инициализация родительского класса
    def __init__(self, **kw):
        super(CalculatorScreen, self).__init__(**kw)

    def on_button_press(self, button):
        # Обработка нажатия кнопки
        if button.text == "C":
            # Сброс значения ввода
            self.input_value.text = "0"
        elif button.text == "DEL":
            # Удаление последнего символа из ввода
            self.input_value.text = self.input_value.text[:-1] if len(self.input_value.text) > 1 else "0"
        elif button.text == "=":
            # Вычисление результата выражения
            try:
                result = str(eval(self.input_value.text))
                # Сохранение выражения и результата в историю
                self.history.append(f"{self.input_value.text} = {result}")
                self.input_value.text = result
            except Exception:
                # Обработка ошибок вычисления
                self.input_value.text = "Error"
        else:
            # Добавление текста кнопки к вводу
            if self.input_value.text == "0":
                self.input_value.text = button.text
            else:
                self.input_value.text += button.text

```

Рисунок 2. Код экрана калькулятора

CalculatorScreen — это экран, на котором отображается сам калькулятор. Он использует два типа контейнеров для размещения виджетов: *BoxLayout* и *GridLayout*.

Свойства:

- *input_value*: свойство, которое ссылается на текстовое поле для отображения текущего ввода. Свойство типа *ObjectProperty* позволяет связывать элементы пользовательского интерфейса с кодом.
- *history*: список для хранения истории вычислений.

Методы:

- *__init__(self, **kw)*: инициализирует класс и вызывает родительский конструктор.
- *on_button_press(self, button)*: Основная функция, которая обрабатывает нажатия кнопок калькулятора.
 - Если нажата кнопка "C", ввод сбрасывается.
 - Если нажата кнопка "DEL", удаляется последний символ ввода.
 - Если нажата кнопка "=", выполняется вычисление введенного выражения с помощью функции *eval()*, и результат

- отображается. В случае ошибки отображается сообщение "Error".
- Для других кнопок (цифры и операторы) добавляется их текст в строку ввода.



Рисунок 3. Внешний вид калькулятора

3. Экран истории расчетов

```

class HistoryScreen(BoxLayout, GridLayout, Screen): 1 usage
    # Свойство для доступа к списку истории
    history_list = ObjectProperty(None)

    def __init__(self, **kw):
        # Инициализация родительского класса
        super(HistoryScreen, self).__init__(**kw)

    def on_enter(self):
        # Обновление истории при входе на экран
        app = App.get_running_app()
        calculator_screen = app.root.get_screen("CalculatorScreen")
        self.update_history(calculator_screen.history)

    def update_history(self, history): 1 usage
        # Очистка текущего списка истории и добавление новых записей
        self.history_list.clear_widgets()
        for entry in history:
            btn = Button(text=entry)
            # Привязка события нажатия кнопки к функции загрузки в калькулятор
            btn.bind(on_press=self.load_to_calculator)
            self.history_list.add_widget(btn)

    def load_to_calculator(self, instance): 1 usage
        # Загружаем выбранный расчет в строку ввода калькулятора
        app = App.get_running_app()
        calculator_screen = app.root.get_screen("CalculatorScreen")
        # Извлекаем только выражение до знака "="
        expression = instance.text.split(' = ')[0]
        calculator_screen.input_value.text = expression
        # Переход на главный экран калькулятора
        app.root.current = "CalculatorScreen"

```

Рисунок 4. Код истории расчетов

HistoryScreen — экран для отображения истории предыдущих вычислений.

Свойства:

- *history_list*: свойство, которое ссылается на список элементов (кнопок), отображающих историю вычислений.

Методы:

- *on_enter(self)*: вызывается, когда пользователь переходит на экран истории. Здесь происходит обновление истории, используя данные с экрана калькулятора.
- *update_history(self, history)*: обновляет список с историей, очищая предыдущие записи и добавляя новые кнопки для каждого вычисления в истории.
- *load_to_calculator(self, instance)*: когда пользователь нажимает на одну из кнопок истории, вычисление загружается в строку ввода калькулятора и происходит переход обратно на экран калькулятора.

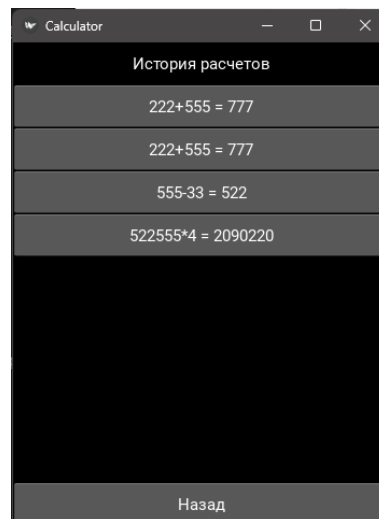


Рисунок 5. Внешний вид истории запросов

4. Менеджер экранов

```
class WinMan(ScreenManager): 1 usage
    # Менеджер экранов для управления переходами между экранами
    pass
```

Рисунок 6. Код менеджера экранов

5. Основной класс приложения

```
class CalculatorApp(App): 1 usage
    def build(self):
        # Установка размера окна приложения
        Window.size = (350, 450)
        return WinMan()

    def on_start(self):
        pass
```

Рисунок 7. Код основного
Класса

CalculatorApp — основной класс приложения. Наследует от класса *App*, который является базой для всех приложений Kivy.

Методы:

- *build(self)*: этот метод вызывается при запуске приложения и отвечает за создание графического интерфейса. Здесь создается экземпляр *WinMan*, который управляет экранами.
- *on_start(self)*: этот метод вызывается сразу после создания приложения, но в данном случае он не содержит дополнительной логики.

6. Запуск приложения

```
# Запуск приложения
if __name__ == '__main__':
    CalculatorApp().run()
```

Рисунок 8. Код запуска приложения

Эта строка запускает приложение. Если файл запускается как основной, создается экземпляр *CalculatorApp* и выполняется метод *run()*, который запускает цикл событий Kivy и отображает приложение на экране.

7. Заключение

Данный код представляет собой полноценное приложение калькулятора с возможностью просмотра истории вычислений. Использование фреймворка Kivy позволяет легко реализовать интерфейс с несколькими экранами и взаимодействием между ними. Приложение демонстрирует основы объектно-ориентированного программирования и обработки событий, что делает его хорошим примером для изучения и дальнейшего использования в образовательных целях.

Теперь создадим файл *Calculator.kv* и пропишем там следующий код.

1. Менеджер экранов

```
<WinMan>:
    # Менеджер экранов, который управляет переходами между экранами
    CalculatorScreen:
        name: "CalculatorScreen" # Имя экрана калькулятора
    HistoryScreen:
        name: "HistoryScreen" # Имя экрана истории расчетов
```

Рисунок 9. Код

<WinMan> — это класс, который является менеджером экранов и управляет переходами между экранами. Он использует механизм переключения между экранами *CalculatorScreen* и *HistoryScreen*.

В *WinMan* указаны имена экранов, которые позволяют в *main.py* обращаться к ним через их идентификаторы.

- *CalculatorScreen* — это экран с калькулятором, где происходят все вычисления.
- *HistoryScreen* — экран для отображения истории расчетов.

2. Экран калькулятора

```
<CalculatorScreen>:
  orientation: "vertical" # Вертикальная ориентация для размещения элементов
  input_value: input_box # Связываем поле ввода с идентификатором input_box
  history: [] # Инициализация пустого списка для хранения истории расчетов

  BoxLayout:
    height: "40dp" # Высота строки ввода
    size_hint_y: None # Отключаем изменение размера по оси Y

    TextInput:
      id: input_box # Идентификатор текстового поля ввода
      text: "0" # Начальное значение
      multiline: False # Запрет на многострочный ввод
      readonly: True # Поле ввода только для чтения
      background_color: 0, 0, 0, 1 # Черный цвет фона
      foreground_color: 1, 1, 1, 1 # Белый цвет текста
      halign: "right" # Выравнивание текста по правому краю
      font_size: 25 # Размер шрифта
```

Рисунок 10. Код

CalculatorScreen — это основной экран калькулятора. Используется вертикальная ориентация для упорядочивания элементов по вертикали.

В этом блоке настраивается поле для ввода чисел:

- *TextInput* с идентификатором *input_box* используется для отображения текста ввода.
- *text: "0"* устанавливает начальное значение в поле ввода, которое будет отображаться по умолчанию.
- Поле ввода настроено как *readonly: True*, что означает, что пользователь не может вводить текст напрямую в это поле, а может только через кнопки калькулятора.
- Цвета фона и текста устанавливаются через параметры *background_color* и *foreground_color*.

3. Кнопки калькулятора

```
# Кнопки для ввода чисел и операций
BoxLayout:
    Button:
        text: "7"
        size_hint_x: 1 # Занимает всю ширину
        on_press: root.on_button_press(self) # Обработка нажатия кнопки
    Button:
        text: "8"
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "9"
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "+"
        size_hint_x: 1
        on_press: root.on_button_press(self)
```

Рисунок 11. Код

Этот блок включает в себя кнопки для ввода чисел и операций. Каждая кнопка имеет текст, который будет отображаться на кнопке.

Все кнопки внутри каждого *BoxLayout* имеют *size_hint_x:1*, что означает, что они будут занимать всю доступную ширину по оси X.

Каждая кнопка вызывает метод *on_button_press(self)* при нажатии, что позволяет обработать нажатие и выполнить соответствующую операцию.

```
BoxLayout:
    Button:
        text: "4"
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "5"
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "6"
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "-"
        size_hint_x: 1
        on_press: root.on_button_press(self)
```

Рисунок 12. Код

Аналогично предыдущему блоку, здесь также добавлены кнопки для ввода чисел и операций.

Операции, такие как минус ("-"), настраиваются через кнопки, а каждое нажатие вызывает обработчик *on_button_press(self)*, который определяет действия при нажатии кнопки.

```
BoxLayout:
    Button:
        text: "1"
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "2"
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "3"
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "*"
        size_hint_x: 1
        on_press: root.on_button_press(self)
```

Рисунок 13. Код

В этом блоке добавлены кнопки для чисел "1", "2", "3", а также для операции умножения ("*").

Каждая кнопка опять же обрабатывается через `on_button_press(self)`.

```
BoxLayout:
    Button:
        text: "C" # Кнопка для сброса ввода
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "0" # Кнопка для ввода нуля
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "DEL" # Кнопка для удаления последнего символа
        size_hint_x: 1
        on_press: root.on_button_press(self)
    Button:
        text: "/" # Кнопка для деления
        size_hint_x: 1
        on_press: root.on_button_press(self)
```

Рисунок 14. Код

В этом блоке добавлены кнопки для операций "C" (сброс), "DEL" (удаление последнего символа), "0" (ввод нуля), и "/" (деление).

Каждая кнопка вызывает метод `on_button_press(self)`, который выполняет соответствующую операцию.

```

BoxLayout:
    Button:
        text: "=" # Кнопка для вычисления результата
        size_hint_x: 0.5 # Занимает половину ширины
        on_press: root.on_button_press(self)
    Button:
        text: "." # Кнопка для ввода десятичной точки
        size_hint_x: 0.25 # Занимает четверть ширины
        on_press: root.on_button_press(self)
    Button:
        text: "История" # Кнопка для перехода к экрану истории
        size_hint_x: 0.25
        on_press:
            root.manager.current = 'HistoryScreen' # Переход на экран истории
            root.manager.transition.direction = "up" # Направление перехода

```

Рисунок 15. Код

В последнем блоке добавлены три кнопки:

= - вычисление результата.

. - десятичная точка.

История — переход на экран истории расчетов с использованием `root.manager.current = 'HistoryScreen'` для смены экрана.

Переход на экран истории осуществляется с эффектом перехода "up" (`root.manager.transition.direction = "up"`).

4. Экран истории расчетов

```

<HistoryScreen>:
    orientation: "vertical" # Вертикальная ориентация для размещения элементов
    history_list: history_list # Связываем список истории с идентификатором history_list

    Label:
        text: "История расчетов" # Заголовок для экрана истории
        size_hint_y: None # Отключаем изменение размера по оси Y
        height: "40dp" # Высота заголовка

    ScrollView:
        GridLayout:
            id: history_list # Идентификатор для списка истории
            cols: 1 # Один столбец
            row_default_height: 40 # Высота каждой строки
            size_hint_y: None # Отключаем изменение размера по оси Y
            height: self.minimum_height # Высота равна минимальной высоте содержимого

```

Рисунок 16. Код

HistoryScreen — экран, на котором отображается история вычислений. Содержит заголовок "История расчетов" и прокручиваемый список с результатами предыдущих вычислений.

Для прокручивания используется *ScrollView*, а для отображения записей истории — *GridLayout* с одним столбцом.

```
Button:
    text: "Назад" # Кнопка для возврата на экран калькулятора
    size_hint_y: None # Отключаем изменение размера по оси Y
    height: "40dp" # Высота кнопки
    on_press:
        root.manager.current = "CalculatorScreen" # Переход на экран калькулятора
```

Рисунок 17. Код

Эта кнопка позволяет пользователю вернуться с экрана истории расчетов обратно на экран калькулятора. Переход осуществляется с помощью `root.manager.current = "CalculatorScreen"`.

5. Заключение

В данном файле описана разметка интерфейса для калькулятора с использованием Kivy. Каждый элемент и кнопка имеют свою четкую роль, и с помощью *BoxLayout* и *GridLayout* удобно организованы все элементы интерфейса, позволяя достичь простоты и функциональности приложения.

4 Выводы

В ходе разработки калькулятора на основе фреймворка Kivy с использованием языка программирования Python был создан удобный и функциональный интерфейс для выполнения математических операций. Работа показала, что Kivy является эффективным инструментом для создания кроссплатформенных графических приложений, включая мобильные устройства и десктопные платформы. В процессе разработки были решены основные задачи, такие как создание интерфейса калькулятора, обработка ввода и вычислений, а также хранение и отображение истории расчетов.

Особенности фреймворка Kivy, такие как гибкость в работе с различными типами компоновок (`'BoxLayout'`, `'GridLayout'`), а также возможность использовать объектно-ориентированное программирование для управления состоянием приложения, обеспечили легкость и удобство в создании интерфейса и обработке пользовательского ввода. Использование стандартных Python-библиотек, таких как `'eval'`, позволило эффективно реализовать вычислительную логику калькулятора, при этом предусмотрена обработка возможных ошибок, что повышает надежность приложения.

Создание двух экранов для калькулятора и истории расчетов позволило улучшить пользовательский опыт, добавив функциональность для просмотра и повторного использования предыдущих расчетов. Это также подчеркивает возможности фреймворка Kivy в области создания многоэкрановых приложений с поддержкой плавных переходов между экранами.

Таким образом, работа демонстрирует возможности Kivy и Python для создания графических приложений с богатым интерфейсом, а также подчеркивает важность разработки удобных и надежных приложений для повседневного использования. Данная разработка может быть использована в качестве базового примера для дальнейшего изучения и создания более сложных приложений, таких как финансовые калькуляторы или

образовательные инструменты для обучения математике и программированию.

Библиографический список

1. Сейдаметов Г. С. Особенности использования языка программирования python в подготовке будущих инженеров-программистов // В сборнике: International Scientific Review. Иваново: Издательство Олимп. С. 50-51.
2. Усеинов Э.А. Объектно-ориентированное программирование в рамках дисциплины «язык программирования python» // Ученые записки Крымского инженерно-педагогического университета. 2012. Выпуск 34. С. 157-160
3. Шкодырев Д.В. Использование языка программирования python 3 и системы компьютерной алгебры сумруна факультативных занятиях по теории чисел // В сборнике: Математическое образование в школе и вузе: теория и практика (mathedu-2015). Материалы V Международной научно-практической конференции. Отв. ред. Н.В. Тимербаева. 2015. С. 287-288
4. Бурков В. В., Бантыш И. В. Использование языка программирования python для решения задач математического моделирования электромеханических систем // В сборнике: Современные проблемы автоматизации и управления в энергетике и машиностроении сборник научных трудов международной научно-практической конференции. Цикл «Автоматизация и управления» кафедры «Технологии машиностроения» (ТМС) Пензенского государственного технологического университета. 2015. С. 226-234
5. Кизянов А.О., Лучанинов Д.В. Разработка игры 2048 на фреймворке Kivy // Постулат. 2016 № 9.