

Geekbrains

**Создание приложения:  
«Генератор дат для календарно-тематического планирования»**

IT-специалист:  
Программист Python Цифровые профессии  
Нохаева С. С.

Элиста  
2024

## Оглавление

Введение .....	3
Назначение и цели создания приложения .....	5
Структура приложения .....	6
Кроссплатформенный фреймворк Python .....	8
Создание первого приложения с помощью Kivy .....	9
Работа с файлами .....	12
Интерфейс Kivy.....	17
Виджеты .....	19
Макеты .....	24
Кнопки.....	26
Пользовательские интерфейсы и навигация .....	28
Панели выбора даты и времени .....	30
Реализация приложения.....	32
Создание класса MainApp .....	33
Создание интерфейса проекта.....	35
Установка учебных периодов и каникул .....	40
Задание расписания уроков .....	42
Задание праздничных дней.....	43
Вывод и сохранение результата .....	44
Заключение .....	46
Список источников и литературы .....	48
Приложения .....	49

## Введение

Рабочая программа относится к важным документам образовательного учреждения, поскольку содержит предмет, цели и задачи обучения на определенном этапе его прохождения и разрабатывается на базе образовательной программы, предложенной Министерством образования на текущий год. Одним из разделов рабочей программы является календарно-тематическое планирование (КТП). Оно не просто рекомендуется Федеральным государственным образовательным стандартом (ФГОС), а является обязательным условием его реализации. КТП предполагает подразделение учебной программы на определенные разделы, которые отражают тематику курса, его содержание и период изучения, кроме того, в календарном плане детально прописывается временной период конкретного курса, каждой его темы и каждого занятия [1].

Многие педагоги сталкиваются с проблемой расстановки дат в календарно-тематическом планировании. Поэтому в рамках обучения на образовательной платформе GeekBrains мной разработано приложение: «Генератор дат для календарно-тематического планирования». Так целью дипломного проекта является создание приложения, которое помогает справиться с этой проблемой.

В последние годы кроссплатформенная технология разработки мобильных и настольных приложений становится все более популярной. Кроссплатформенный подход позволяет создавать приложения для различных платформ с одной кодовой базой, что экономит время и деньги, и избавляет разработчиков от ненужных усилий. Фреймворк Kivy с библиотекой KivyMD позволяет создавать кроссплатформенные приложения, способные работать на любом устройстве (настольный компьютер, планшет, смартфон, миникомпьютер) и в любой операционной системе (Windows, Linux, MacOS, Android, iOS, Raspberry Pi) [2]. Здесь аббревиатура MD означает Material Design. Material Design – это стандарт, созданный Google, которому нужно придерживаться при разработке приложений для Android и iOS. Kivy предоставляет язык проектирования, специально

ориентированный на простой и масштабируемый дизайн графического интерфейса пользователя. Язык позволяет легко отделить дизайн интерфейса от логики приложения, придерживаясь принципа разделения задач [3].

В своей дипломной работе я ставлю перед собой следующие задачи:

1. изучение фреймворка Kivy и инструментов его работы;
2. изучение языка разметки KV;
3. разработка приложения.

## Назначение и цели создания приложения

Своей целью я ставлю разработку приложения, которое на входе запрашивает:

- начальную и конечную дату промежутка времени учебного года;
- начальные и конечные даты промежутков времени каникул;
- количество уроков в дни недели по предмету;
- нерабочие праздничные дни;

и на выходе:

- генерирует все даты из выбранного временного промежутка нужных дней недели;
- при необходимости сохраняет их в текстовый документ.

Целевая аудитория – это, прежде всего, работники образования: организаторы учебного процесса и педагогические работники.

## Структура приложения

Структура приложения, которую будет видеть перед собой пользователь, будет выглядеть следующим образом:

- окно «Учебный период и каникулы» представлено в приложении 1. В этом окне пользователь нажатием на кнопки «Выбрать даты начала и окончания года», «Выбрать даты начала и окончания каникул» открывает окно календаря и выбирает желаемый промежуток дат. По завершению введения данных нажимает кнопку «Вперёд» и попадает в следующее окно.

- Окно «Количество уроков» представлено в приложении 2. В этом окне пользователь вводит в текстовые поля количество уроков в соответствующие дни недели. По завершению введения данных нажимает кнопку «Вперёд» и попадает в следующее окно.

- Окно «Праздничные дни» представлено в приложении 3. В этом окне пользователь нажатием на кнопки «Выбрать дату» открывает окно календаря и выбирает праздничную дату, которая выводится в текстовом поле. Нажимая кнопку несколько раз, пользователь может ввести нужное количество праздничных нерабочих дней, которые не войдут в результирующий список дат уроков. По завершению введения данных нажимает кнопку «Вперёд» и попадает в следующее окно.

- Окно «Результат» представлено в приложении 4. В этом окне при нажатии на кнопке «Вывод результатов» в текстовое поле выводится результирующий список дат. При необходимости сохранения выведенных данных можно нажатием кнопки «Сохранить» сохранить данные в текстовый файл. Для завершения работы приложения нужно нажать на кнопку «Выход».

Навигация между окнами осуществляется в соответствии с вышеописанной структурой и отображается внизу экрана в виде кнопок управления: «Вперёд», «Назад» и «Выход».

## Кроссплатформенный фреймворк Python

Современным людям бывает просто необходимо иметь выход в Интернет со своего мобильного устройства. Средства сотовой связи обеспечивают подключение к сети Интернет с планшета или смартфона практически в любой точке вне дома или офиса, а специально созданные мобильные приложения позволяют решать как деловые задачи, так и выполнять развлекательные функции. Мобильные приложения действительно захватили нашу жизнь. Почти каждый день мы используем такие средства общения как WhatsApp и Viber, LinkedIn, обучающие приложения и игры.

Различные компании через мобильные приложения могут рассказать о своих товарах и услугах, найти потенциальных партнеров и клиентов, организовать продажу товаров. Рядовые пользователи взаимодействуют с торговыми точками, используют интернет-банкинг, общаются через мессенджеры, получают государственные услуги.

Для разработки мобильных приложений существует множество языков программирования, причем они позволяют создавать мобильные приложения для устройств, работающих либо только под Android, либо под iOS. Но из этих инструментальных средств хочется выделить связку: Python, фреймворк Kivy и библиотека KivyMD.

Kivy – это фреймворк Python, который упрощает создание кроссплатформенных приложений, способных работать в Windows, Linux, Android, OSX, iOS и мини компьютерах типа Raspberry Pi. Это популярный пакет для создания графического интерфейса на Python, который набирает большую популярность благодаря своей простоте в использовании, хорошей поддержке сообщества и простой интеграции различных компонентов.

Библиотека KivyMD построена на основе фреймворка Kivy. Это набор виджетов Material Design (MD) для использования с Kivy. Данная библиотека



предлагает достаточно элегантные компоненты для создания интерфейса – UI (user interface – пользовательский интерфейс), в то время как программный код на Kivu используется для написания основных функций приложения, например, доступ к ресурсам Интернет, обращение к элементам мобильного устройства, таким как видеокамера, микрофон, GPS приемник и т. п.

Используя Python и Kivu можно создавать действительно универсальные приложения, которые из одного программного кода будут работать: на настольных компьютерах (OS X, Linux, Windows);

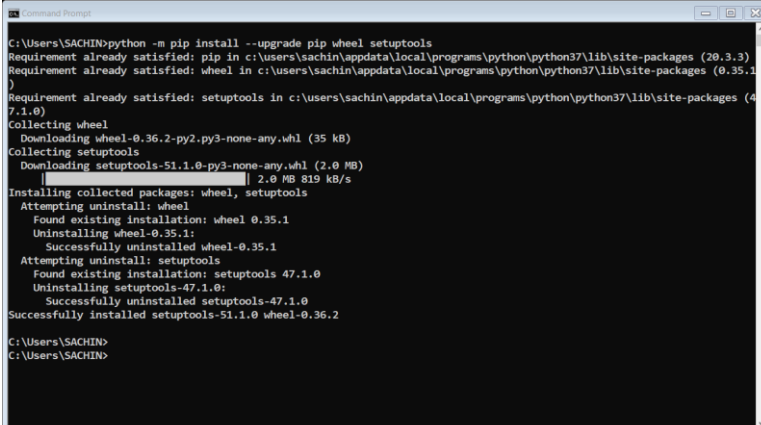
- на устройствах iOS (iPad, iPhone);
  - на Android-устройствах (планшеты, смартфоны);
  - на любых других устройства с сенсорным экраном, поддерживающие TUIO (Tangible User Interface Objects).
- Kivu дает возможность написать программный код один раз и запустить его на совершенно разных платформах [3].

### Создание первого приложения с помощью Kivu

Установка в Windows:

- Шаг 1: Обновите pip и wheel перед установкой kivu, введя эту команду в cmd-

*python -m pip install --upgrade pip wheel setuptools*



```
C:\Users\SACHIN>python -m pip install --upgrade pip wheel setuptools
Requirement already satisfied: pip in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (20.3.3)
Requirement already satisfied: wheel in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (0.35.1)
Requirement already satisfied: setuptools in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (47.1.0)
Collecting wheel
  Downloading wheel-0.36.2-py2.py3-none-any.whl (35 kB)
Collecting setuptools
  Downloading setuptools-51.1.0-py3-none-any.whl (2.0 MB)
    | 2.0 MB 819 kB/s
Installing collected packages: wheel, setuptools
  Attempting uninstall: wheel
    Found existing installation: wheel 0.35.1
    Uninstalling wheel-0.35.1:
      Successfully uninstalled wheel-0.35.1
  Attempting uninstall: setuptools
    Found existing installation: setuptools 47.1.0
    Uninstalling setuptools-47.1.0:
      Successfully uninstalled setuptools-47.1.0
Successfully installed setuptools-51.1.0 wheel-0.36.2

C:\Users\SACHIN>
C:\Users\SACHIN>
```

- Шаг 2: Установите зависимости-

```
python -m pip install docutils pygments pypiwin32 kivy.deps.sdl2  
kivy.deps.glew
```

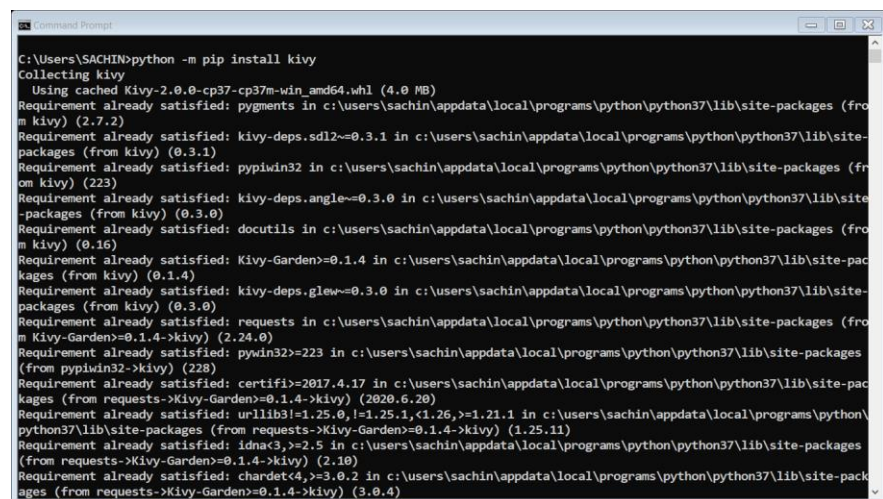
```
python -m pip install kivy.deps.gstreamer
```

```
python -m pip install kivy.deps.angle
```

- Шаг 3: Установите kivy.

```
python -m pip install kivy
```

Результатом будет что-то вроде приведенного ниже изображения:



```

C:\Users\SACHIN>python -m pip install kivy
Collecting kivy
  Using cached Kivy-2.0.0-cp37-cp37m-win_amd64.whl (4.0 MB)
Requirement already satisfied: docutils in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from kivy) (2.7.2)
Requirement already satisfied: pygments in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from kivy) (2.7.2)
Requirement already satisfied: pypiwin32 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from kivy) (223)
Requirement already satisfied: kivy-deps.sdl2==0.3.1 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from kivy) (0.3.1)
Requirement already satisfied: kivy-deps.glew==0.3.0 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from kivy) (0.3.0)
Requirement already satisfied: docutils in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from kivy) (0.16)
Requirement already satisfied: Kivy-Garden>=0.1.4 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from kivy) (0.1.4)
Requirement already satisfied: kivy-deps.gstreamer==0.3.0 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from kivy) (0.3.0)
Requirement already satisfied: requests in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from Kivy-Garden>=0.1.4->kivy) (2.24.0)
Requirement already satisfied: pypiwin32==223 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from pypiwin32->kivy) (223)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from requests->Kivy-Garden>=0.1.4->kivy) (2020.6.20)
Requirement already satisfied: urllib3<1.25.0, >=1.25.1, <1.26, >=1.21.1 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from requests->Kivy-Garden>=0.1.4->kivy) (1.25.11)
Requirement already satisfied: idna<3, >=2.5 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from requests->Kivy-Garden>=0.1.4->kivy) (2.10)
Requirement already satisfied: chardet<4, >=3.0.2 in c:\users\sachin\appdata\local\programs\python\python37\lib\site-packages (from requests->Kivy-Garden>=0.1.4->kivy) (3.0.4)

```

Установка в Linux:

- Шаг 1: Добавьте PPA, введя эту команду в терминале-

```
sudo add-apt-repository ppa:kivy-team/kivy
```

- Шаг 2: Обновите список пакетов с помощью вашего менеджера пакетов-

```
sudo apt-get update
```

- Шаг 3: Установите Kivy

*sudo apt-get install python3-kivy*

Создание приложения с помощью kivy состоит из трех этапов:

1. Наследует класс приложения Kivy, который представляет окно для наших виджетов
2. Создайте метод `build()`, который будет отображать содержимое виджетов.
3. И, наконец, вызов метода `run()` [18].

## Работа с файлами

Файл — это набор данных, сохраненный в виде последовательности битов на компьютере. Информация хранится в куче данных (структура данных) и имеет название «имя файла» (*filename*).

В Python существует два типа файлов:

- текстовые
- бинарные

Текстовые файлы – это файлы с человекочитаемым содержимым. В них хранятся последовательности символов, которые понимает человек. Блокнот и другие стандартные редакторы умеют читать и редактировать этот тип файлов. Текст может храниться в двух форматах: (*.txt*) — простой текст и (*.rtf*) — «формат обогащенного текста».

В бинарных файлах данные отображаются в закодированной форме (с использованием только нулей (0) и единиц (1) вместо простых символов). В большинстве случаев это просто последовательности битов. Они хранятся в формате *.bin*.

Любую операцию с файлом можно разбить на три крупных этапа:

1. открытие файла
2. выполнение операции (запись, чтение)
3. закрытие файла

Открытие файла - метод `open()`.

В Python есть встроенная функция `open()`. С ее помощью можно открыть любой файл на компьютере. Технически Python создает на его основе объект.

Синтаксис, следующий:

```
f = open(file_name, access_mode)
```

где, *file\_name* = имя открываемого файла

*access\_mode* = режим открытия файла. Он может быть: для чтения, записи и т. д. По умолчанию используется режим чтения (*r*), если другое не указано. Режимы открытия файла представлены в Приложении 5

Следующий код используется для его открытия.

```
f = open('example.txt', 'r') # открыть файл из рабочей директории в ре-  
жиме чтения
```

```
fp = open('C:/xyz.txt', 'r') # открыть файл из любого каталога
```

Закрывание файла - метод `close()`.

После открытия файла в Python его нужно закрыть. Таким образом освобождаются ресурсы и убирается мусор. Python автоматически закрывает файл, когда объект присваивается другому файлу.

Существуют следующие способы:

Способ №1

Проще всего после открытия файла закрыть его, используя метод `close()`.

```
f = open('example.txt', 'r')
```

```
# работа с файлом
```

```
f.close()
```

После закрытия этот файл нельзя будет использовать до тех пор, пока заново его не открыть.

## Способ №2

Также можно написать *try/finally*, которое гарантирует, что если после открытия файла операции с ним приводят к исключениям, он закроется автоматически. Без него программа завершается некорректно.

Вот как сделать это исключение:

```
f = open('example.txt', 'r')
```

```
try:
```

```
    # работа с файлом
```

```
finally:
```

```
    f.close()
```

Файл нужно открыть до инструкции *try*, потому что если инструкция *open* сама по себе вызовет ошибку, то файл не будет открываться для последующего закрытия.

Этот метод гарантирует, что если операции над файлом вызовут исключения, то он закроется до того, как программа остановится.

## Способ №3

Инструкция *with*

Еще один подход — использовать инструкцию *with*, которая упрощает обработку исключений с помощью инкапсуляции начальных операций, а также задач по закрытию и очистке.

В таком случае инструкция *close* не нужна, потому что *with* автоматически закроет файл.

Вот как это реализовать в коде.

```
with open('example.txt') as f:
```

```
    # работа с файлом
```

Чтение и запись файлов в Python

В Python файлы можно читать или записывать информацию в них с помощью соответствующих режимов.

Функция *read()*. Функция *read()* используется для чтения содержимого файла после открытия его в режиме чтения (r).

Синтаксис

```
file.read(size)
```

где, *file* = объект файла, *size* = количество символов, которые нужно прочитать. Если не указать, то файл прочитается целиком.

Функция *readline()*.

Функция *readline()* используется для построчного чтения содержимого файла. Она используется для крупных файлов. С ее помощью можно получать доступ к любой строке в любой момент.

Функция *write()*. Функция *write()* используется для записи в файлы Python, открытые в режиме записи.

Если попытаться открыть файл, которого не существует, в этом режиме, тогда будет создан новый.

Синтаксис

*file.write(string)*

Переименование файлов в Python

Функция `rename()`. Функция `rename()` используется для переименовывания файлов в Python. Для ее использования сперва нужно импортировать модуль `os`.

Синтаксис следующий.

*import os*

*os.rename(src,dest)*

где, *src* = файл, который нужно переименовать, *dest* = новое имя файла

Методы файла в Python приведены в приложении 6. [19]



## Интерфейс Kivy

При использовании фреймворка Kivy программный код для создания элементов пользовательского интерфейса можно писать на Python, а можно для этих целей использовать специальный язык. В литературе можно встретить разное обозначение этого языка: язык kivy, язык KV, KV.

Язык KV обеспечивает решение следующих задач:

- создавать объекты на основе базовых классов Kivy;
- формировать дерево виджетов (создавать контейнеры для размещения визуальных элементов и указывать их расположение на экране);
- задавать свойства для виджетов;
- естественным образом связывать свойства виджетов друг с другом;
- связывать виджеты с функциями, в которых обрабатываются различные события.

Язык KV позволяет достаточно быстро и просто создавать прототипы программ и гибко вносить изменения в пользовательский интерфейс. Это также обеспечивает при программировании отделение логики приложения от пользовательского интерфейса.

Есть два способа загрузить программный код на KV в приложение.

- По соглашению об именах. В этом случае Kivy ищет файл с расширением». kv» и с тем же именем, что и имя базового класса приложения в нижнем регистре, за вычетом символов «App». Например, если базовый класс приложения имеет имя MainApp, то для размещения кода на языке KV нужно использовать файл с именем main. kv. Если в этом файле задан корневой виджет, то он

будет использоваться в качестве основы для построения дерева виджетов приложения.

– С использованием специального модуля (компоненты) Builder можно подключить к приложению программный код на языке KV либо из строковой переменной, либо из файла с любым именем, имеющем расширение `kv`. Если в данной строковой переменной или в этом файле задан корневой виджет, то он будет использоваться в качестве основы для построения дерева виджетов приложения.

У компоненты Builder есть два метода для загрузки в приложение кода на языке KV:

– `Builder.load_file('path/name_file.kv')` – если код на языке KV подгружается из файла (здесь `path` – путь к файлу, `name_file.kv` – имя файла);

– `Builder.load_string(kv_string)` – если код на языке KV подгружается из строковой переменной (`kv_string` – имя строковой переменной). [6]

Интерфейс пользователя в приложениях на Kivy строится на основе виджетов. Виджеты Kivy можно классифицировать следующим образом:

- UX-виджеты, или видимые виджеты (они отображаются в окне приложения, и пользователь взаимодействует с ними);
- виджеты контейнеры или «макеты» (они не отображаются в окне приложения и служат контейнерами для размещения в них видимых виджетов);
- сложные UX-виджеты (комбинация нескольких виджетов, обеспечивающая совмещение их функций);
- виджеты поведения (контейнеры, которые обеспечивают изменение поведения находящихся в них элементов);

- диспетчер экрана (особый виджет, который управляет сменой экранов приложения) [3].

Для создания интерфейса мы должны создать и назвать этот файл так же, как и класс, используя строчные буквы и расширение `.kv`. Kivy автоматически свяжет имя класса и имя файла.

Внутри этого `.kv`-файла необходимо создать несколько экранов и указать макеты приложения, включая такие элементы, как, кнопки, метки, формы и т.д. Макеты в Kivy бывают разных типов, но выполняют одну и ту же функцию: это контейнеры, используемые для расположения виджетов в соответствии с выбранным макетом [7].

## Виджеты

### Виджет MDLabel

Класс MDLabel (метка, этикетка) предназначен для отображения в окне приложения текста различных размеров и цветов, а также иконок (подкласс MDIcon) [3].

MDLabel имеет следующие параметры:

- `text` - текст, который мы хотим поместить на метку
- `halign` - позиция, в которой мы хотим разместить метку.
- `theme_text_color` - тема для таких цветов текста, как пользовательский, основной, вторичный, подсказка или ошибка
- `text_color`- если параметр `theme_text_color` является пользовательским, мы можем назначить цвет текста кортежу RGB.
- `font_style` - шрифт, заголовки [8].

Пример реализации вывода текста на основе MDLabel в окне ввода учебных периодов и каникул:

```
MDLabel:
    text: "Учебный период и каникулы"
    font_size: 40
    halign: 'center'
    size_hint_y: None
    height:50
```

## Виджет TextInput

Виджет TextInput предоставляет поле для редактируемого обычного текста. Поддерживаются Unicode, многострочность, навигация курсором, выделение и функции буфера обмена [11].

TextInput предоставляет различные обработчики событий для отслеживания взаимодействия пользователя. Например, on\_text\_validate событие запускается, когда пользователь заканчивает вводить текст и нажимает клавишу Enter:

```
def build(self):

    text_input = TextInput(text='Enter text and press Enter...')

    text_input.bind(on_text_validate=self.on_enter)

    return text_input
```

TextInput позволяет, также, настраивать его внешний вид, изменяя такие атрибуты, как размер шрифта (font\_size), цвет текста (foreground\_color) и цвет фона (background\_color).

Мы можем ограничить максимальную длину текста, вводимого в виджете TextInput. Это полезно, когда мы хотим ввести ограничение на количество символов, например, max\_length: 10 [12].

Мы в нашем приложении будем использовать виджет TextInput для ввода количества уроков в дни недели:

```
TextInput:
    id:monday_count
    text: "0"
    hint_text: "Введите количество уроков"
```

## Виджет MDTextField

Класс MDTextField позволяет создать поле без рамок для ввода текста. Такое поле с параметрами по умолчанию выделено в окне приложения нижней линией подчеркивания. Текстовое поле MDTextField унаследовано от класса TextInput фреймворка Kivy. Следовательно, большинство параметров и все события класса TextInput также доступны и в классе MDTextField [3].

### Атрибуты класса MDTextField:

- hint\_text - для печати текста подсказки в текстовом поле (принимает строку);
- helper\_text - для печати вспомогательного текста в текстовом поле (принимает строку);
- helper\_text\_mode - для изменения режима helper\_text (принимает on\_focus и persistent);
- pos\_hint - для изменения положения кнопки (принимает значения в формате словаря, например  
pos\_hint : {"center\_x":.5,"center\_y":.5}).
- icon\_right - название значка, который будет напечатан в правой части текстового поля.
- size\_hint\_x - для изменения размера поля по оси x (принимает значения типа float и None);
- width - используется, когда значение size\_hint\_x равно None (принимает числовые значения);
- color\_mode - для изменения цвета текстового поля (принимает значения accent, primary)

- `line_color_focus` - для изменения цвета строки текстового поля при фокусировке (принимает значения в формате `rgb`);
- `line_color_normal` - чтобы изменить цвет строки текстового поля, когда он нормальный (принимает значения в формате `rgb`);
- `icon_right_color` - для изменения цвета правой иконки (принимает значения в формате `rgb`);
- `multiline` - принимает многострочный ввод от пользователя (принимает `True`, `False`);
- `mode` - для изменения режима текстового поля (значениями могут быть прямоугольник, заливка);
- `fill_color` - используется, когда используется режим заливки и изменения цвета фона кнопки (принимает значения в формате `rgb`) [13].

## Виджет ScrollView

Кроме виджетов для позиционирования в Kivy есть еще два особых класса, которые обеспечивают создание контейнеров для скроллинга вложенных в них элементов:

- `ScrollView` – для организации вертикального и горизонтального скроллинга;
- `Carousel` – для организации горизонтального скроллинга [3].

В нашем приложении мы используем `ScrollView`.

Виджет `ScrollView` обеспечивает создание окна, в котором можно прокручивать видимые элементы интерфейса. `ScrollView` принимает только один дочерний элемент и включает его в область прокрутки. Таким вложенным элементом может быть один из контейнеров (например, `BoxLayout`, `GridLayout` и т.п.) который содержит множество других визуальных элементов. Прокрутку можно выполнять и в вертикальном, и в горизонтальном направлении, это зависит от значений свойств `scroll_x` и `scroll_y`. Данный виджет анализирует характер

прикосновения для того, чтобы определить, хочет ли пользователь прокрутить контент, или прикосновением активизировать какую-либо функцию, связанную с визуальным элементом. При этом пользователь не может делать и то, и другое одновременно. Чтобы определить, является ли касание жестом прокрутки, используются следующие свойства:

- `scroll_distance`: минимальное расстояние для перемещения (по умолчанию 20 пикселей);
- `scroll_timeout`: максимальный период времени (по умолчанию 55 миллисекунд).

Если выполняется касание с перемещением по пикселям на дистанцию «`scroll_distance`» в течение периода «`scroll_timeout`», оно распознается как жест прокрутки, и начинается скроллинг содержимого окна. В противном случае фиксируется простое касание (без перемещения), запускается обработка события простого касания, и скроллинг окна не происходит. Значение по умолчанию для этих параметров можно изменить в файле конфигурации:

```
[widgets]
```

```
scroll_timeout = 250
```

```
scroll_distance = 20
```

Виджет `ScrollView` используем для вывода результатов работы приложения:

```
ScrollView:
  MDTextField:
    id: dates
    multiline: True
    halign: 'center'
    size_hint_y: None
    height:50
```

## Макеты

Макеты — это контейнеры, используемые для упорядочивания виджетов определенным образом.

`AnchorLayout`: виджеты могут быть привязаны к ‘верху’, ‘низу’, ‘левому’, ‘правому’ или ‘центру’.

`BoxLayout`: виджеты расположены последовательно, либо в "вертикальной", либо в ‘горизонтальной’ ориентации.

`FloatLayout`: виджеты практически не ограничены.

`RelativeLayout`: дочерние виджеты расположены относительно макета.

`GridLayout`: виджеты расположены в виде сетки, определяемой свойствами `rows` и `cols`.

`PageLayout`: используется для создания простых многостраничных макетов таким образом, чтобы можно было легко переключаться с одной страницы на другую с помощью границ.

`ScatterLayout`: виджеты расположены аналогично `RelativeLayout`, но их можно перемещать, поворачивать и масштабировать.

`StackLayout`: виджеты расположены в *lr-tb* (слева направо, затем сверху вниз) *tb-lr*.

Компонента `MDBoxLayout` является невидимым элементом интерфейса, это некий контейнер, в котором располагаются другие, видимые элементы интерфейса (кнопки, строки, метки и т.п.) [2]. `MDBoxLayout` размещает виджеты либо вертикально, то есть один поверх другого, либо горизонтально, то есть один



за другим. Если мы не укажем никакой подсказки о размере, дочерние виджеты разделят размер своего родительского виджета поровну или соответственно [10].

Свойства:

- `orientation` (ориентация) – задает ориентацию макета и по умолчанию имеет значение «horizontal». Может быть «vertical» или «horizontal»;
- `padding` (отступ) – задает величину отступа виджета от границ его родительского контейнера (в пикселях).
- `spacing` (интервал) – расстояние между дочерними элементами, находящимися внутри контейнера (в пикселях).

Контейнер имеет следующий набор свойств, которые по умолчанию имеют значения False:

- `adaptive_height` – разрешить позиционирование элементов по высоте и адаптировать это положение к размерам контейнера;
- `adaptive_width` – разрешить позиционирование элементов по ширине и адаптировать это положение к размерам контейнера;
- `adaptive_size` – разрешить позиционирование элементов и по высоте, и ширине и адаптировать это положение к размерам контейнера.
- `pos_hint` – расположение кнопки [3].

Класс `MDGridLayout` служит для размещения виджетов в ячейках таблицы. С его помощью создается контейнер (или макет), в котором будут размещаться другие виджеты со своими свойствами. Данный контейнер представляет собой таблицу, состоящую из столбцов и строк. Каждый элемент, вложенный в этот контейнер, занимает одну ячейку этой таблицы. Сам контейнер `MDGridLayout` можно разместить в любом месте окна приложения [2].

Мы не можем явно разместить виджеты в определенном столбце / строке. Каждому дочернему элементу присваивается определенная позиция,

автоматически определяемая конфигурацией макета и дочерним индексом в списке дочерних элементов. Сеточное описание должно содержать, по крайней мере, входные ограничения, т. е. столбцы и строки. Если мы не укажем для него `cols` или `rows`, макет выдаст вам исключение.

- Начальный размер задается свойствами `col_default_width` и `row_default_height`. Мы можем принудительно установить размер по умолчанию, установив свойство `col_force_default` или `row_force_default`. Это заставит макет игнорировать дочерние свойства `width` и `size_hint` и использовать размер по умолчанию.
- Чтобы настроить размер отдельного столбца или строки, используйте `cols_minimum` или `rows_minimum`.
- Необязательно указывать как строки, так и столбцы, это зависит от требований. Мы можем предоставить либо обе, либо любую другую информацию соответственно.

## Кнопки

Кнопка — это метка с соответствующими действиями, которые запускаются при нажатии кнопки (или отпускании после щелчка / касания). Мы можем добавить функции за кнопкой и оформить кнопку.

Кнопка `Button` имеет ряд свойств, которые позволяют задать надпись на кнопке, параметры шрифта, и запустить реакцию на события или изменение состояния:

- `text` — надпись на кнопке;
- `font_size` — размер шрифта;
- `color` — цвет шрифта;
- `font_name` — имя файла с пользовательским шрифтом (.ttf).
- `on_press` — событие, возникает, когда кнопка нажата;

- `on_release` – событие, возникает, когда кнопка отпущена;
- `on_state` – состояние (изменяется тогда, когда кнопка нажата или отпущена) [3].

Класс `MDRaisedButton` позволяет создавать традиционные кнопки. Эта кнопка похожа на кнопку с надписью, она отличается тем, что для нее можно установить цвет фона [2].

#### Атрибуты класса `MDRaisedButton`

- `text` - текст, который будет напечатан в кнопке (принимает строку)
- `pos_hint` - для размещения кнопки в определенном положении (принимает значения с плавающей точкой)
- `theme_text_color` : Для изменения цвета значка (значения `Primary`, `Secondary`, `Подсказка`, `Ошибка` и `Custom`)

Примечание: если значение `theme_text_color` является пользовательским, мы можем использовать атрибут `text_color` для изменения цвета.

- `text_color` - для изменения цвета значка (принимает значения цвета в формате `rgb`)
- `высота` - для создания тени кнопки (принимает числовое значение)
- `font_size` - для увеличения размера шрифта кнопки (принимает числовое значение)
- `md_bg_color` - для изменения цвета фона кнопки используется название цвета в формате `RGB`.
- `on_press` - действие, которое должно выполняться при нажатии кнопки
- `on_release` - действие, которое будет выполнено при отпуске кнопки [9].

В свойствах, связанных с событиями (`on_press`, `on_release`), можно указать `lambda` функцию, то есть обработать эти события любой внешней функцией.

Пример создания `MDRaisedButton`, которую будем использовать для выбора даты начала и окончания учебного года:

```
MDRaisedButton:
    id: btn0
    text: "Выбрать даты начала и окончания учебного года"
    md_bg_color: 60/255, 174/255, 60/255, 122/255
    halign: 'center'
    on_release: root.input_study_period('btn0')
```

## Пользовательские интерфейсы и навигация

### Виджет `ScreenManager`

`ScreenManager` - это виджет, который используется для управления несколькими экранами вашего приложения. `ScreenManager` по умолчанию отображает только один экран одновременно и использует базу переходов для переключения с одного экрана на другой. Поддерживаются множественные переходы.

Изменение переходов:

По умолчанию доступно несколько переходов, таких как:

- `NoTransition` – мгновенно переключает экраны без анимации
- `SlideTransition` – перемещение экрана внутрь / наружу в любом направлении
- `CardTransition` – новый экран накладывается на предыдущий или старый соскальзывает с нового, в зависимости от режима
- `SwapTransition` – реализация перехода iOS подкачки
- `FadeTransition` – шейдер для затемнения экрана при входе / выходе
- `WipeTransition` – шейдер для стирания экранов справа налево

- `FallOutTransition` – шейдер, при котором старый экран "падает" и становится прозрачным, открывая новый экран за ним.
- `RiseInTransition` – шейдер, при котором новый экран поднимается из центра экрана, переходя из прозрачного в непрозрачный.

Можно переключать переходы, изменяя свойство

`ScreenManager.transition:`

```
sm = ScreenManager(transition=FadeTransition())
```

В нашем приложении перемещение между экранами мы будем осуществлять с помощью экранного менеджера, импорт которого осуществим командой:

```
from kivy.uix.screenmanager import ScreenManager, Screen
```

Значения цветов в Kivy

Значения цветов в Kivy не являются обычными значениями RGB — они нормализованы. Для понимания нормализации цвета необходимо знать, что распределение значений цвета обычно зависит от освещенности. Оно зависит от таких факторов, как условия освещения, эффект объектива и другие.

Чтобы избежать этой проблемы, Kivy использует соглашение `(1, 1, 1)`. Это представление Kivy для RGB `(255, 255, 255)`. Для преобразования обычных значений RGB в соглашение Kivy нам нужно разделить все ваши значения на `255`. Таким образом мы получим значения в диапазоне от 0 до 1 [5].

Пример выбора цвета:

```
MDRaisedButton:
```

```
id: btn1
text: "Выбрать даты начала и окончания каникул"
halign: 'center'
md_bg_color: 60/255, 174/255, 60/255, 122/255
on_release: root.input_study_period('btn1')
```

## Панели выбора даты и времени

В KivyMD предоставлены следующие классы для создания сборных панелей:

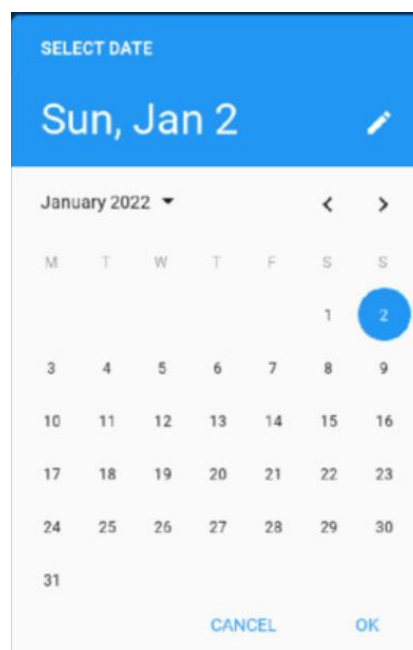
- `MDDTimePicker` – панель для задания времени;
- `MDDDatePicker` – панель для задания даты;
- `MDDThemePicker` – панель для смены темы интерфейса приложения.

В нашем проекте мы будем использовать для выбора учебных периодов и каникул, а также, для ввода праздничных нерабочих дней.

В базовом классе имеется две функции:

- `def on_save` – для обработки события `on_save` (получить выбранную дату) и передачи этого значения метке `date_label`;
- `def on_cancel` – для обработки события нажатия кнопки `cancel` в диалоговом окне с элементом `MDDDatePicker`.

После открытия диалогового окна `MDDDatePicker` по умолчанию установлена текущая дата.



Кнопки с символами «<», «>» позволяют пролистать месяцы текущего года, а кнопка рядом с указанием текущего года позволяет открыть (и скрыть) окно для выбора другого года. После нажатия на кнопку «ОК» выбранная дата будет возвращена в приложение

Теперь вернемся к настройкам окна `MDDatePicker`. Имеется возможность установить любую начальную дату, например:

```
date_dialog = MDDatePicker (year=2016, month=4, day=12)
```

После этого при открытии окна текущая дата будет заменена на ту, которая установлена в последней строке. Кроме того, в `MDDatePicker` можно выбрать интервал дат. Для выбора диапазона дат используется параметр

## Реализация приложения

Структура проекта, которую будет видеть перед собой пользователь, будет выглядеть следующим образом:

- окно ввода учебных периодов и каникул;
- окно ввода количества уроков в дни недели;
- окно ввода праздничных нерабочих дней;
- окно вывода результата.

При работе с проектом, в первую очередь, создадим виртуальное окружения. Таким образом, мы изолируем проект от других проектов, хранящихся на этом же ПК. Кроме того изменения внутри проекта не влияют на "эталонную" версию Python, которую установили.

Для создания виртуального окружения в каталоге нашего проект выполним команду:

```
python -m venv venv
```

Таким образом, внутри проекта будет создана папка *venv* в которой будет находиться локальная версия Python с которой мы будем работать. Далее следует активировать виртуальное окружение командой:

```
venv\Scripts\activate.
```

В папке *venv* будут храниться все зависимости проекта.

Создание приложения с помощью Kivy состоит из трех этапов:

- Наследование класса приложения Kivy, который представляет окно для наших виджетов
- Создание метод *build()*, который будет отображать содержимое виджетов.
- Вызов метода *run()* [4].

Командой:

```
pip install kivy
```

устанавливаем фреймворк Kivy в виртуальное окружение.



После установки Kivy установим KivyMD (дополнение к фреймворку Kivy) командой:

```
pip install kivymd
```

Рассмотрим схему жизненного цикла приложения Kivy, представленную в приложении 7.

Как вы можете видеть, нашей точкой входа в наше приложение является метод `run()`, и в нашем случае это “`MainApp().run()`” [17].

## Создание класса `MainApp`

Создадим класс, который будет определять наше приложение. Я назову его `MainApp`. Этот класс будет наследовать класс `MdApp` из KivyMD. Поэтому необходимо импортировать приложение, добавив:

```
from kivymd.app import MdApp.
```

В класс `MainApp` необходимо добавить метод `build`. Импортируем его, добавив:

```
from kivy.lang import Builder.
```

Чтобы вернуть пользовательский интерфейс, мы будем использовать функцию `build` [5]. Возвращаем его в виде экрана. Для этого необходимо импортировать `ScreenManager` и `Screen` с помощью строки:

```
from kivy.uix.screenmanager import ScreenManager, Screen.
```

```
class MainApp(MdApp):  
    def build(self):  
        self.screen = Builder.load_file(kv)  
        return self.screen
```

Для достижения целей нашего приложения введём глобальные переменные:

```
kv = 'date_generator.kv'  
school_days = []  
week_days = []  
result_school_days = []
```

Так как мы собираемся работать более чем с одним окном, то нам нужно создать класс, который будет управлять навигацией между этими окнами. Этот класс необходимо будет унаследовать от *ScreenManager* [7]:

```
class WindowManager(ScreenManager):  
    pass
```

Для каждого окна, которое мы хотим создать, нам нужно создать пустой класс, который наследуется от класса *Screen*. Описание логики этих классов опишем в следующих пунктах.

```
class StudyPeriodsWindow(Screen):  
    pass  
    Saglara *  
class WeekdaysWindow(Screen):  
    pass  
    Saglara *  
class PublicHolidaysWindow(Screen):  
    pass  
    Saglara *  
class ResultWindow(Screen):  
    pass  
    Saglara  
class WindowManager(ScreenManager):  
    pass
```

## Создание интерфейса проекта

Для реализации пользовательского интерфейса создадим .kv файл.

В верхней части нашего файла определим, какие окна будут находиться внутри нашего *ScreenManager*, сделав отступ для них под тегом класса *ScreenManager*:

```
WindowManager:
    StudyPeriodsWindow:
    WeekdaysWindow:
    PublicHolidaysWindow:
    ResultWindow:
```

Для осуществления навигации нам нужно добавить атрибут *name* для каждого из наших окон:

```
<StudyPeriodsWindow>:
    name: "periods"

<WeekdaysWindow>:
    name: "weekdays"

<PublicHolidaysWindow>:
    name: "publicHolidays"

<ResultWindow>:
    name: "result"
```

Во всех наших окнах основным будет макет *MDBoxLayout*, в который будут помещаться дочерние макеты и виджеты.

```
<StudyPeriodsWindow>:
    name: "periods"

    MDBoxLayout:
        orientation: 'vertical'
        padding: 50
        spacing: 30
```

Установим вертикальную ориентацию, отступы и интервалы.

Установим соответствующие заголовки для наших. Для этого во всех макеты MDBoxLayout добавим MDLabel и установим свойство «text» соответственно названию каждого окна. Например, зададим заголовок первого окна:

```
MDLabel:
    text: "Учебный период и каникулы"
    font_size: 40
    halign: 'center'
    size_hint_y: None
    height: 50
```

Чтобы обеспечить навигацию между нашими окнами, добавим во все макеты MDBoxLayout наших окон кнопки MDRaisedButton. Для каждой кнопки добавим надпись («Назад» или «Вперёд») и событие *on\_release*, которое обеспечит переход в следующее окно, либо в предыдущее окно в зависимости от того, для чего она будет предназначена. Также можем выбрать направление, в котором мы хотим, чтобы окно перемещалось при выполнении перехода. Продемонстрируем это на примере окна ввода праздничных дней:

```
MDBoxLayout:

    MDRaisedButton:
        text: "Назад"
        on_release:
            app.root.current = "weekdays"
            root.manager.transition.direction = "right"

    MDRaisedButton:
        text: "Вперёд"
        on_release:
            app.root.current = "result"
            root.manager.transition.direction = "left"
```

Окно «Учебный период и каникулы».

В первом окне добавим в контейнер MDBoxLayout дочерний компонент MDGridLayout, который в свою очередь состоит из кнопок MDRaisedButton и меток MDLabel:

```
MDBoxLayout:
    MDLabel:
        text: "Учебный период и каникулы"
    MDGridLayout:
        cols: 2
        MDRaisedButton:
            text: "Выбрать даты начала и окончания учебного года"
        MDLabel:
            text: "гггг-мм-дд. - гггг-мм-дд."
        MDRaisedButton:
            text: "Выбрать даты начала и окончания каникул"
        MDLabel:
            text: "гггг-мм-дд. - гггг-мм-дд."
        MDRaisedButton:
            text: "Выбрать даты начала и окончания каникул"
        MDLabel:
            text: "гггг-мм-дд. - гггг-мм-дд."
        MDRaisedButton:
            text: "Выбрать даты начала и окончания каникул"
        MDLabel:
            text: "гггг-мм-дд. - гггг-мм-дд."
    MDRaisedButton:
        text: "Вперёд"
```

Кнопки MDRaisedButton предназначены для ввода промежутков учебного периода и каникул. Метки MDLabel – для вывода выбранных промежутков на экран.

Всем кнопам и меткам мы зададим свойство id, с помощью которого мы будем однозначно определять каждый виджет. Каждой кнопке пропишем событие *on\_release*, для вызова функции, с помощью которой мы будем сохранять выбранную дату. Пример пары MDRaisedButton и MDLabel для ввода учебного периода.

```
MDRaisedButton:
    id: btn0
    text: "Выбрать даты начала и окончания учебного года"
    md_bg_color: 60/255, 174/255, 60/255, 122/255
    halign: 'center'
    on_release: root.input_study_period('btn0')

MDLabel:
    id: date1
    text: "yyyy-мм-дд. - yyyy-мм-дд."
    halign: 'center'
```

Окно «Количество уроков».

Во втором окне добавим в контейнер MDBoxLayout дочерний компонент MDGridLayout, который в свою очередь состоит из пары: строка ввода TextInput и метка MDLabel для каждого дня недели. Например:

```
MDLabel:
    text: "Понедельник"
    halign: 'center'

TextInput:
    id: monday_count
    text: "0"
    hint_text: "Введите количество уроков"
```

Окно «Праздничные дни».

В третьем окне добавим в контейнер MDBoxLayout дочерний компонент MDBoxLayout, который в свою очередь состоит из кнопки MDRaisedButton и текстового поля MDTextField. Кнопке MDRaisedButton пропишем свойство *on\_release*, для вызова функции *input\_public\_holidays()*, с помощью которой мы будем выбирать, сохранять и выводить на печать праздничные даты.

```
MDRaisedButton:
    id: btn1
    md_bg_color:60/255, 174/255, 60/255, 122/255
    pos_hint: {"center_x":.5, "center_y":.5}
    text: "Выбрать дату"
    halign: 'center'
    on_release: root.input_public_holidays()
```

Даты будут выводиться в поле MDTextField, для которого установлена полоса прокрутки и свойства:

```
ScrollView:
    MDTextField:
        id: dates
        multiline: True
        halign: 'center'
        size_hint_y: None
        height:50
```

Окно «Результат».

В последнем окне добавим в контейнер MDBoxLayout кнопку MDRaisedButton и текстовое поле MDTextField. Кнопке MDRaisedButton пропишем свойство *on\_release*, для вызова функции *output\_of\_results()*, с помощью которой мы будем выводить на печать результат работы приложения.

```
MDRaisedButton:
    text: "Вывод результатов"
    font_size: 30
    size: 100, 100
    size_hint: None, None
    md_bg_color:60/255, 174/255, 60/255, 122/255
    pos_hint: {"center_x":.5}
    on_release: root.output_of_results()
```

Даты занятий будут выводиться в поле MDTextField, для которого установлена полоса прокрутки и свойства:

```

ScrollView:
    MDTextField:
        id: dates
        multiline: True
        halign: 'center'
        size_hint_y: None
        height:50

```

## Установка учебных периодов и каникул

Для определения дат календарно-тематического планирования необходимо в первую очередь ввести даты начала и окончания учебного года, а затем удалить периоды каникул.

Встроенные средства выбора дат идеально подходят для навигации по датам как в ближайшем будущем, так и в прошлом, так и в отдаленном будущем, поскольку они предоставляют несколько способов выбора дат. Для осуществления этого действия мы будем, при нажатии кнопки, вызывать средство выбора времени с помощью *MDDatePicker()*, который позволяет нам открыть календарь и выбрать дату, а затем вернуть её метку *MDLabel* [14]. Но, предварительно, импортируем пакет *MDDatePicker*:

```
from kivymd.uix.picker import MDTimePicker
```

и введём глобальные переменные, которые нам необходимы:

```

school_days = []
week_days = []
result_school_days = []

```

где:

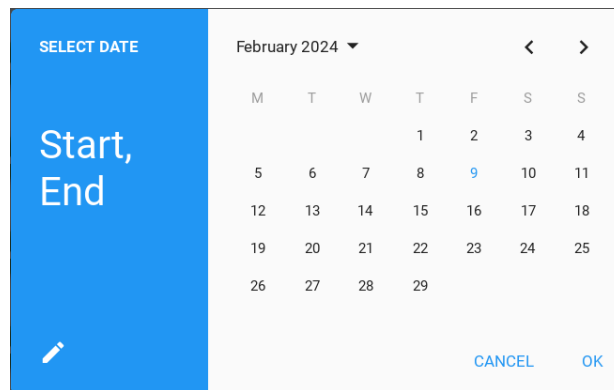
*school\_days* – список всех рабочих дней за исключением каникул;

*week\_days* – список количества уроков в соответствующие дни недели;



*result\_school\_days* – результирующий список дат.

Затем в классе *StudyPeriodsWindow()* создадим функцию *input\_study\_period*, которая будет выводить окно календаря для выбора диапазона дат:



Для выбора диапазона дат используем параметр *mode="range"*:

```
date_dialog = MDDatePicker(mode='range')
```

Для сохранения и вывода в метке выбранного периода, помощью функции *bind*, вызовем функцию *on\_save()*.

```
class StudyPeriodsWindow(Screen):  
  
    new *  
    def input_study_period(self, x):  
        date_dialog = MDDatePicker(mode='range')  
        date_dialog.bind(on_save=partial(self.on_save, x), on_cancel=self.on_cancel)  
        date_dialog.open()
```

В функции *on\_save()* объявляем глобальную переменную *school\_days* и формируем список рабочих дат при нажатии кнопки с *id = btn0*. Одним из аргументов функции является идентификатор кнопки, которая была нажата. Мы его передаём, чтобы затем с помощью оператора выбора *match-case* определить в какую из имеющихся меток записать выбранный период. Даты периода каникул будут удаляться из списка рабочих дней *school\_days*. Для отображения средства выбора времени мы используем метод *open()* [15]:

```

def on_save(self, x, instance, value, date_range):
    global school_days

    match x:
        case 'btn0':
            self.ids.date1.text = f'{str(date_range[0])} - {str(date_range[-1])}'
            school_days = date_range
        case 'btn1':
            self.ids.date2.text = f'{str(date_range[0])} - {str(date_range[-1])}'
            school_days = [x for x in school_days if x not in date_range]
        case 'btn2':
            self.ids.date3.text = f'{str(date_range[0])} - {str(date_range[-1])}'
            school_days = [x for x in school_days if x not in date_range]
        case 'btn3':
            self.ids.date4.text = f'{str(date_range[0])} - {str(date_range[-1])}'
            school_days = [x for x in school_days if x not in date_range]

```

## Задание расписания уроков

Для того, чтобы составить список дат проведения учебных занятий, нам нужно знать расписание занятий по дням недели. Поэтому в следующем классе *WeekdaysWindow(Screen)* мы создаём функцию *input\_working\_days\_of\_the\_week()*, в которой формируем расписание уроков. Для этого мы запрашиваем у пользователя количество уроков в соответствующие дни недели.

```

class WeekdaysWindow(Screen):
    def input_working_days_of_the_week(self):
        global week_days

        week_days = [int(self.ids.monday_count.text), int(self.ids.tuesday_count.text),
                     int(self.ids.wednesday_count.text), int(self.ids.thursday_count.text),
                     int(self.ids.friday_count.text), int(self.ids.saturday_count.text)]

        return week_days

```

Пользователь вводит в соответствующие дням недели строки ввода *TextInput* количество уроков в этот день. Каждая строка ввода имеет свой идентификатор, ссылаясь на который, мы считываем именно то количество уроков, которое соответствует выбранному дню. Затем мы из полученных данных формируем список *week\_days*. Каждое число в этом списке является количеством

уроков в соответствующий день недели. Функция *input\_working\_days\_of\_the\_week()* возвращает список *week\_days*.

## Задание праздничных дней

Следующим этапом нашей задачи является удаление праздничных нерабочих дней, по которым занятия не будут проводиться. Для этого мы в следующем классе, *PublicHolidaysWindow(Screen)*, создадим функцию *input\_public\_holidays(self)*, которая запускается нажатием на кнопку «Выбрать дату» и используя средство выбора времени *MDDatePicker()* будет открывать окно календаря для ввода даты очередного праздничного нерабочего дня:

```
def input_public_holidays(self):
    date_dialog = MDDatePicker()
    date_dialog.bind(on_save=self.on_save, on_cancel=self.on_cancel)
    date_dialog.open()
```

Для сохранения даты используем функцию *on\_save(self, instance, value, date\_range)*:

```
def on_save(self, instance, value, date_range):
    global school_days

    self.ids.dates.text = self.ids.dates.text + f"\n{str(value)}"
    school_days = [x for x in school_days if x not in [value]]
```

которая выводит выбранную праздничную дату в текстовое поле *MDTextField*, по идентификатору *dates* и удаляет её из списка всех рабочих дней. Таким образом, на данном этапе у нас имеются:

- список рабочих дней, за исключением каникул и праздничных нерабочих дней;
- список, содержащий недельное расписание уроков.

## Вывод и сохранение результата

Последним этапом нашего проекта является вывод и сохранение результатов. Для этого в классе `ResultWindow(Screen)`, реализуем функцию *output\_of\_results*, которая вызывается кнопкой «Вывод результатов»:

```
def output_of_results(self):
    global week_days
    global school_days
    global result_school_days

    for single_date in school_days:
        for i in range(len(week_days)):
            if single_date.weekday() == i:
                for j in range(week_days[i]):
                    self.ids.dates.text = self.ids.dates.text \
                        + f"\n{single_date.strftime('%Y-%m-%d')}"
                    result_school_days.append(single_date)
```

Данная функция создаёт результирующий список дат (*result\_school\_days*) и выводит его на печать в поле *MDTextField*, используя списки: *week\_days*, *school\_days*, которые были сформированы в результате работы предыдущих этапов.

В цикле мы для каждой даты из списка рабочих дней *school\_days* определяем соответствующий ей день недели. Затем печатаем текущую дату в поле вывода *MDTextField* такое количество раз, какое количество уроков в соответствующий ей день недели указано в списке *week\_days*. Далее текущая дата добавляется в результирующий список, который можно будет, при необходимости, сохранить в текстовый файл.

Сохранение результатов работы в текстовый файл происходит нажатием на кнопку «Сохранить». Все созданные даты записываются в файл с расширением *.txt*. Для этого мы создали функцию *save\_to\_file*:

```
def save_to_file(self):  
    global result_school_days  
  
    MyFile = open('dates.txt', 'w')  
    for element in result_school_days:  
        MyFile.write(str(element))  
        MyFile.write('\n')  
    MyFile.close()
```

С помощью функции *open()* создадим текстовый файл *dates.txt* и сохраним его в рабочей директории, используя режим *w* - только для записи, который создаст новый файл, если не найдет с указанным именем. Далее в цикле мы производим построчную запись дат в файл и закрываем его используя метод *close()*.

## Заключение

Kivy – это фреймворк Python, который упрощает создание кроссплатформенных приложений, способных работать в Windows, Linux, Android, OSX и iOS. Самое лучшее в Kivy то, что он работает лучше, чем кроссплатформенные альтернативы HTML5. Это популярный пакет для создания графического интерфейса на Python, который набирает большую популярность благодаря своей простоте в использовании, хорошей поддержке сообщества и простой интеграции различных компонентов. Чтобы изучить все возможности Kivy мало одного проекта, с ним хочется работать и дальше, изучая другие библиотеки и средства реализации различных функциональностей. Kivy разработан для сенсорных устройств и обеспечивает поддержку сенсорного ввода и жестов, что упрощает создание естественных и интуитивно понятных пользовательских интерфейсов.

В результате своей работы я полностью реализовала задачи, которые ставила перед собой на старте работы над проектом:

1. изучила структуру фреймворка Kivy, познакомилась и применила на практике инструменты работы ним;
2. изучила особенности языка разметки KV, возможности создания пользовательских классов и объектов;
3. разработала приложение «Генератор дат для КТП»

В ходе проекта была реализована следующая структура приложения:

- окно «Учебный период и каникулы» для ввода учебного периода и каникул;
- окно «Количество уроков» для ввода расписания уроков;
- окно «Праздничные дни» для выбора праздничных дат
- окно «Результат» для вывода результирующего список дат.

Подводя итоги проектной работы, можно сделать вывод, что поставленная цель была достигнута. Данное приложение уже может быть полезно для расстановки дат для календарно-тематического планирования рабочих программ. Можно дополнительно выполнить компиляцию приложения, используя инструмент Buildozer, для Android и использовать его на телефоне.

Ссылка на проект в удаленном репозитории github:  
[https://github.com/SaglaraNokhaeva/Diplom\\_DG](https://github.com/SaglaraNokhaeva/Diplom_DG)

## Список источников и литературы

1. [https://spravochnick.ru/pedagogika/kalendarno-tematicheskoe\\_planirovanie/](https://spravochnick.ru/pedagogika/kalendarno-tematicheskoe_planirovanie/)
2. Фреймворк Kivy — Документация Kivy 2.3.0
3. Разработка кроссплатформенных мобильных и настольных приложений на Python Практическое пособие Анатолий Постолиит, 2022
4. <https://www.geeksforgeeks.org/creating-your-first-application-using-kivy/>
5. <https://pythonist.ru/kivy-python-framework/>
6. <https://фб2.рф/razrabotka-krossplatformennyh-mobilnyh-i-nbsp-nastolnyh-prilogheniy-na-nbsp-python-prakticheskoe-p-67257131/read/part-4>
7. <https://kivy.org/doc/stable/api-kivy.uix.screenmanager.html>
8. <https://www.geeksforgeeks.org/python-add-label-to-a-kivy-window/>
9. <https://pythondevelopers14.blogspot.com/2020/07/mdraisedbutton.html>
10. <https://www.geeksforgeeks.org/python-boxlayout-widget-in-kivy/>
11. <https://www.geeksforgeeks.org/python-textinput-widget-in-kivy/>
12. <https://markscodenotes.com/mastering-textinput-in-pythons-kivy-a-user-friendly-guide-with-code-examples/>
13. <https://pythondevelopers14.blogspot.com/2020/08/mdtextfield.html>
14. <https://www.geeksforgeeks.org/create-time-picker-using-kivymd/>
15. <https://kivymd.readthedocs.io/en/latest/components/date picker/index.html>
16. <https://pythonru.com/osnovy/fajly-v-python-vvod-vyvod>
17. Kivy Documentation. *Release 2.2.0*. The Kivy Developers
18. <https://www.geeksforgeeks.org/introduction-to-kivy/?ref=lbp>
19. <https://pythonru.com/osnovy/fajly-v-python-vvod-vyvod>



The screenshot shows a web application window titled 'Main'. The main heading is 'Учебный период и каникулы'. Below the heading, there are four rows of input fields. Each row consists of a green button with white text and a corresponding date format placeholder. The buttons are labeled: 'Выбрать даты начала и окончания учебного года', 'Выбрать даты начала и окончания каникул', 'Выбрать даты начала и окончания каникул', and 'Выбрать даты начала и окончания каникул'. The date format placeholders are 'ГГГГ-ММ-ДД. - ГГГГ-ММ-ДД.', 'ГГГГ-ММ-ДД. - ГГГГ-ММ-ДД.', 'ГГГГ-ММ-ДД. - ГГГГ-ММ-ДД.', and 'ГГГГ-ММ-ДД. - ГГГГ-ММ-ДД.' respectively. At the bottom center, there is a large purple button with white text labeled 'Вперёд'.

Учебный период и каникулы

Выбрать даты начала и окончания учебного года ГГГГ-ММ-ДД. - ГГГГ-ММ-ДД.

Выбрать даты начала и окончания каникул ГГГГ-ММ-ДД. - ГГГГ-ММ-ДД.

Выбрать даты начала и окончания каникул ГГГГ-ММ-ДД. - ГГГГ-ММ-ДД.

Выбрать даты начала и окончания каникул ГГГГ-ММ-ДД. - ГГГГ-ММ-ДД.

Вперёд

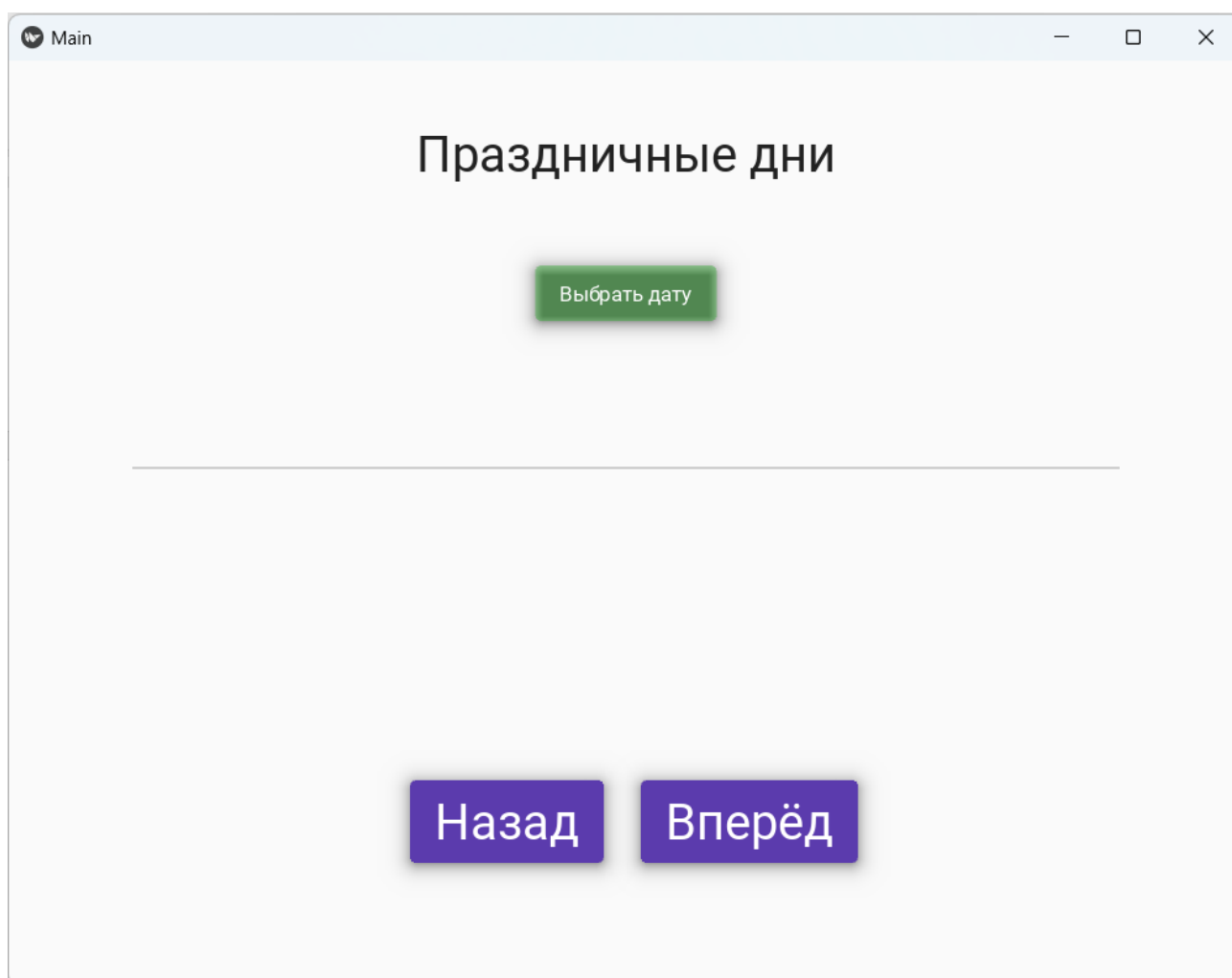
Main

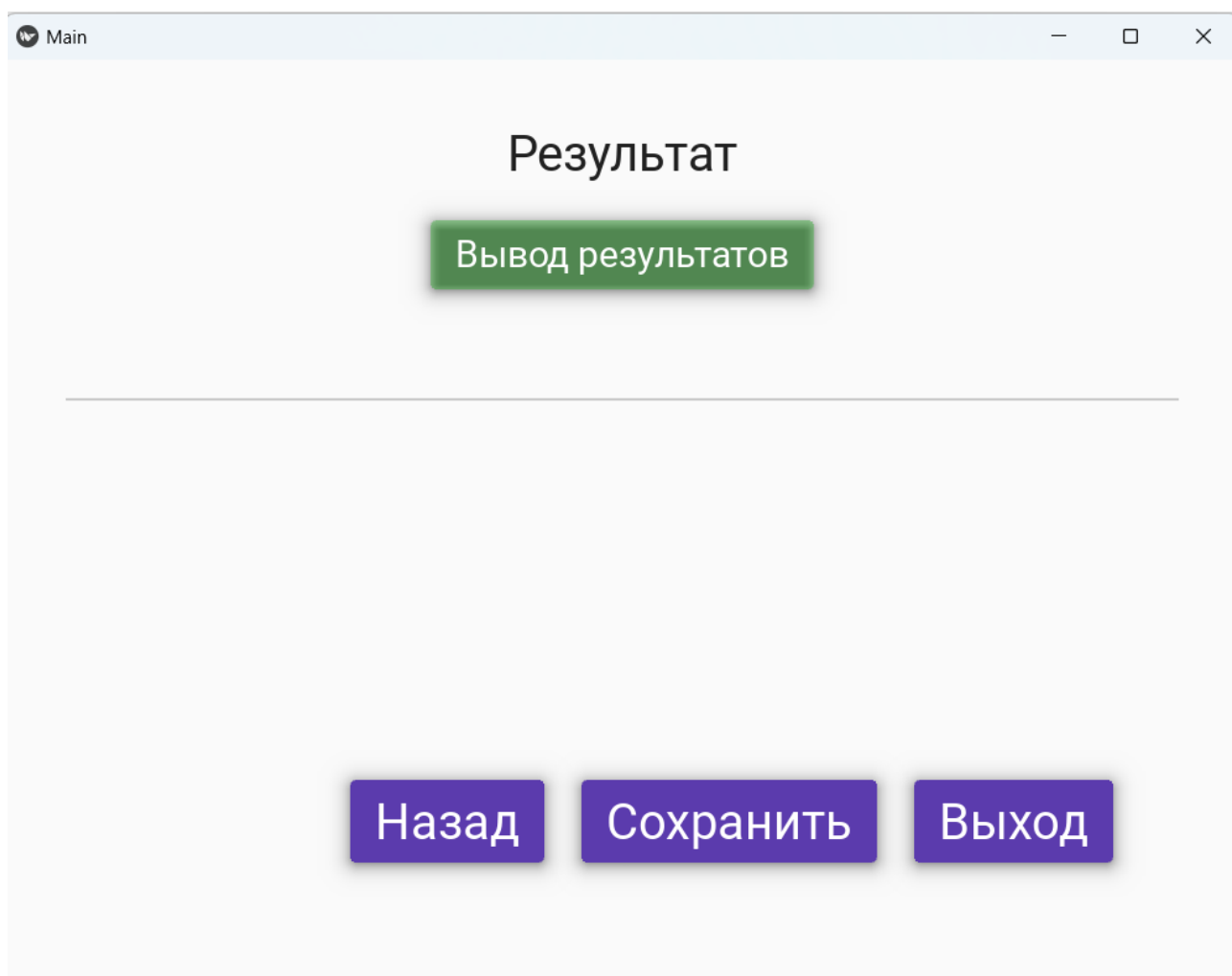
## Количество уроков

Понедельник	0
Вторник	0
Среда	0
Четверг	0
Пятница	0
Суббота	0

Назад

Вперёд





Режим	Описание
r	Только для чтения.
w	Только для записи. Создаст новый файл, если не найдет с указанным именем.
rb	Только для чтения (бинарный).
wb	Только для записи (бинарный). Создаст новый файл, если не найдет с указанным именем.
r+	Для чтения и записи.
rb+	Для чтения и записи (бинарный).
w+	Для чтения и записи. Создаст новый файл для записи, если не найдет с указанным именем.
wb+	Для чтения и записи (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
a	Откроет для добавления нового содержимого. Создаст новый файл для записи, если не найдет с указанным именем.
a+	Откроет для добавления нового содержимого. Создаст новый файл для чтения записи, если не найдет с указанным именем.
ab	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
ab+	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для чтения записи, если не найдет с указанным именем.

Метод	Описание
<code>file.close()</code>	закрывает открытый файл
<code>file.fileno()</code>	возвращает целочисленный дескриптор файла
<code>file.flush()</code>	очищает внутренний буфер
<code>file.isatty()</code>	возвращает True, если файл привязан к терминалу
<code>file.next()</code>	возвращает следующую строку файла
<code>file.read(n)</code>	чтение первых n символов файла
<code>file.readline()</code>	читает одну строчку строки или файла
<code>file.readlines()</code>	читает и возвращает список всех строк в файле
<code>file.seek(offset[,whene])</code>	устанавливает текущую позицию в файле
<code>file.seekable()</code>	проверяет, поддерживает ли файл случайный доступ. Возвращает True, если да
<code>file.tell()</code>	возвращает текущую позицию в файле
<code>file.truncate(n)</code>	уменьшает размер файл. Если n указала, то файл обрезается до n байт, если нет — до текущей позиции
<code>file.write(str)</code>	добавляет <u>строку</u> str в файл
<code>file.writelines(sequence)</code>	добавляет последовательность строк в файл

Схема жизненного цикла приложения Kivy.

