Лекция 4

Приложение с UI для обработки изображений

- ✓ Пользовательский интерфейс
- ✓ Работа с изображением

В Python существует несколько популярных библиотек для создания пользовательских интерфейсов (UI). Вот список наиболее известных:

1. Tkinter

Встроенная библиотека для создания графических интерфейсов. Проста в использовании, но имеет ограниченные возможности для создания современных интерфейсов.

Преимущества:

- ✓ Входит в стандартную библиотеку Python (не требует установки).
- ✓ Подходит для простых приложений.

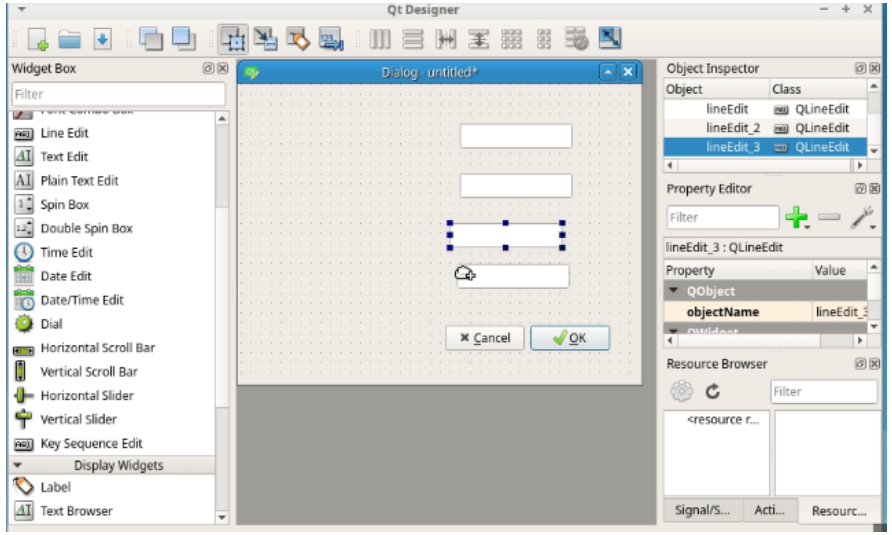
Недостатки:

Ограниченный набор виджетов и стилей.

Документация: https://docs.python.org/3/library/tkinter.html

2. PyQt

PyQt5 является одним из самых популярных фреймворков Python для графического интерфейса. Пакет PyQt построен на платформе Qt, которая представляет собой кроссплатформенную среду, используемую для создания различных приложений на разных платформах.



Преимущества:

✓ Мощный и гибкий фреймворк.

✓ Поддержка современных UI-элементов.

✓ Возможность создания сложных интерфейсов.

Недостатки:

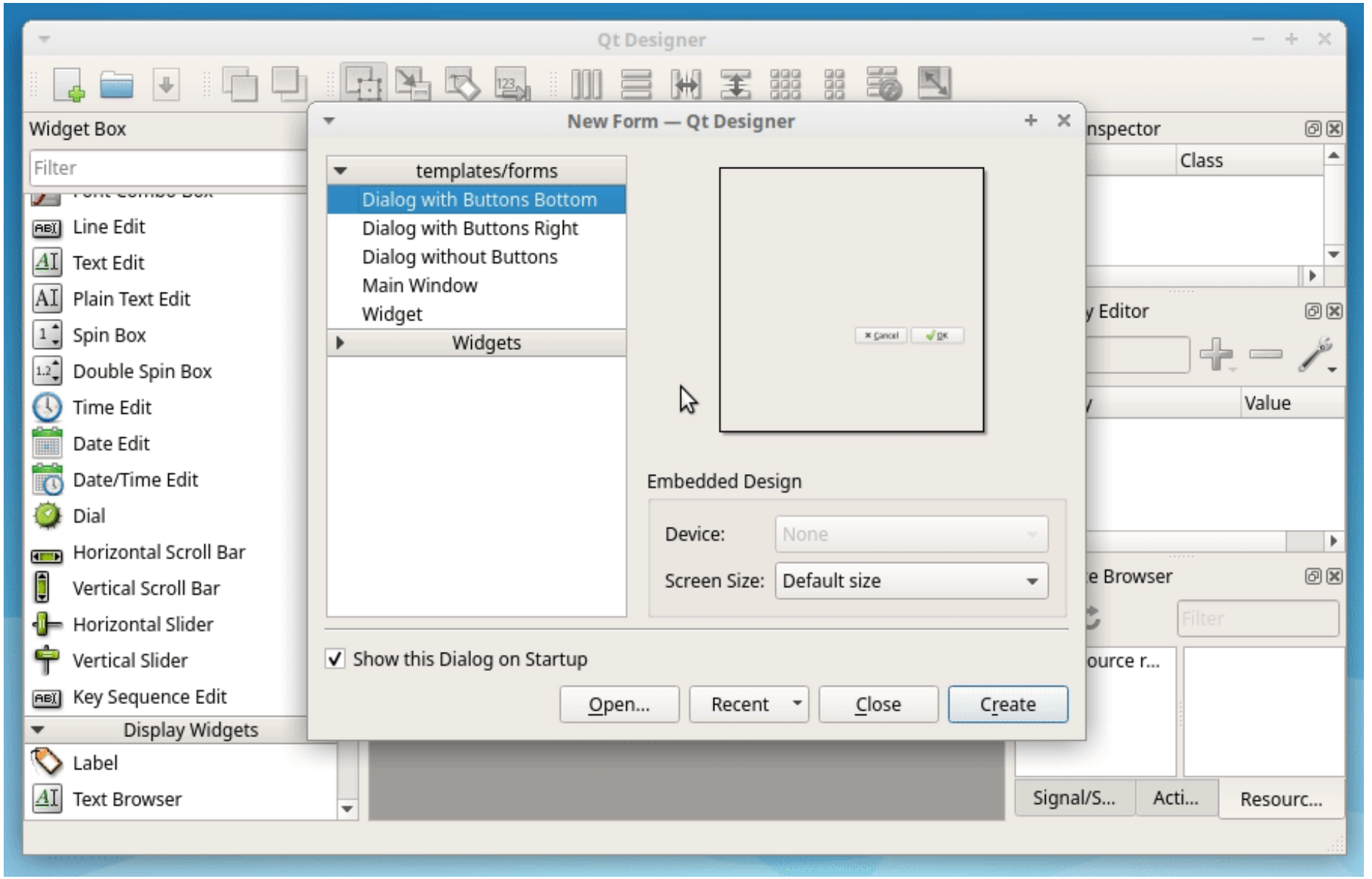
✓ Требует установки.

✓ Более сложный в освоении по сравнению с Tkinter.

Документация:

PyQt: https://www.riverbankcomputing.com/software/pyqt/

PyQt5


Qt Designer
Widget Box
Filter
Line Edit
Text Edit
Plain Text Edit
Spin Box
Double Spin Box
Time Edit
Date Edit
Date/Time Edit
Dial
Horizontal Scroll Bar
Vertical Scroll Bar
Horizontal Slider
Vertical Slider
Key Sequence Edit
Display Widgets
Label
Text Browser
New Form — Qt Designer
templates/forms
Dialog with Buttons Bottom
Dialog with Buttons Right
Dialog without Buttons
Main Window
Widget
Widgets
Embedded Design
Device:
None
Screen Size:
Default size
Show this Dialog on Startup
Open...
Recent
Close
Create
Class
Value
Signal/S...
Acti...
Resourc...

3. Kivy

Библиотека для создания мультитач-приложений. Подходит для создания мобильных приложений и приложений с сенсорным управлением. Обеспечивает поддержку различных платформ, таких как Windows, Mac, Linux, Android и iOS.

Преимущества:

✓ Кроссплатформенность (Windows, macOS, Linux, Android, iOS).

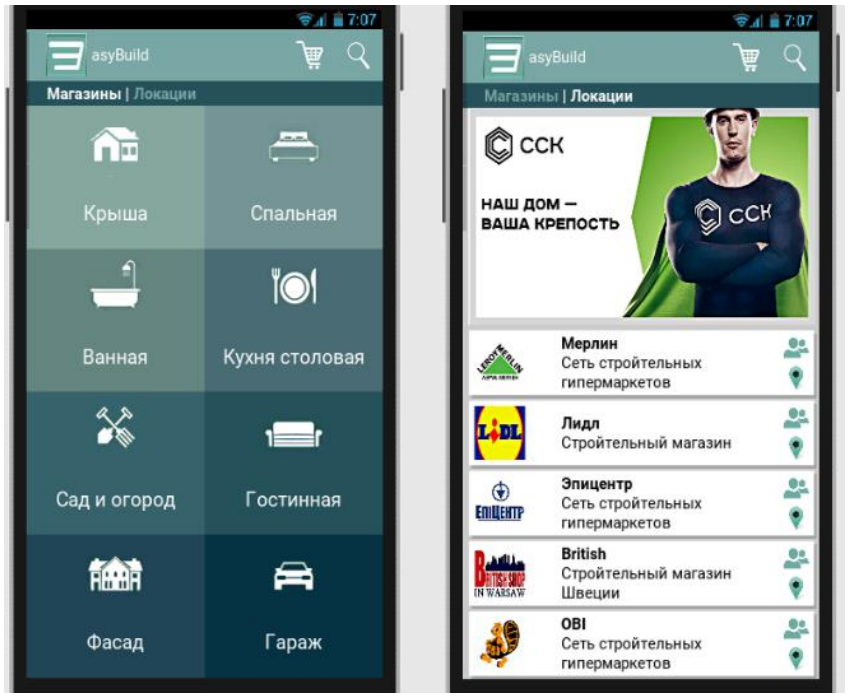
✓ Поддержка мультитач и современных интерфейсов.

Недостатки:

✓ Требует установки.

✓ Не подходит для декстопных приложений.

Документация: https://kivy.org/



4. PySimpleGUI

Обертка над Tkinter, Qt, wxPython и другими библиотеками. Упрощает создание интерфейсов для новичков.

Преимущества:

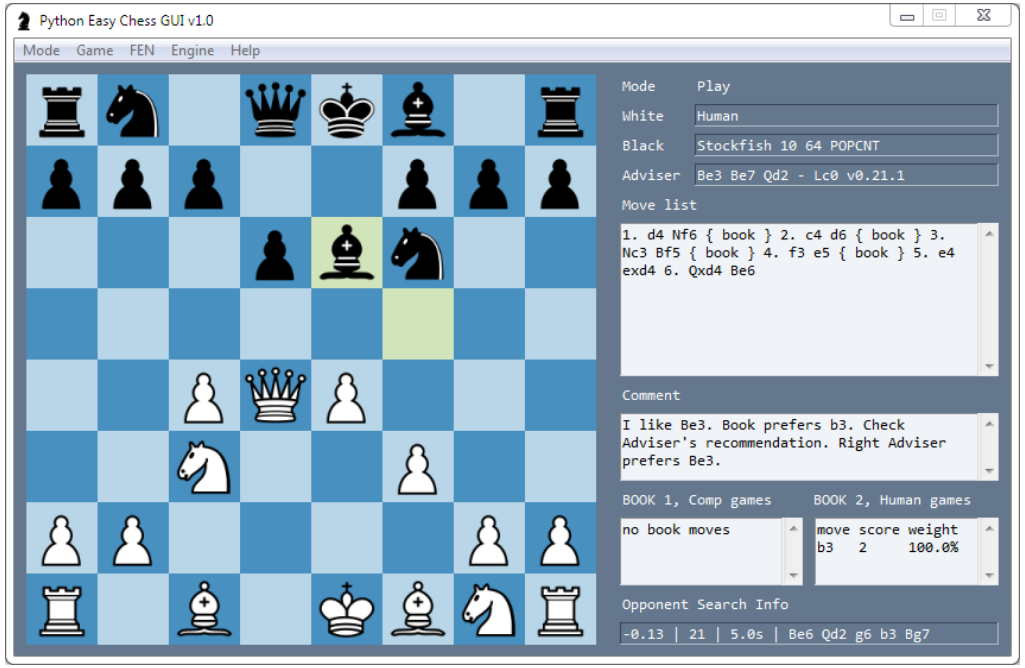
✓ Простота использования.

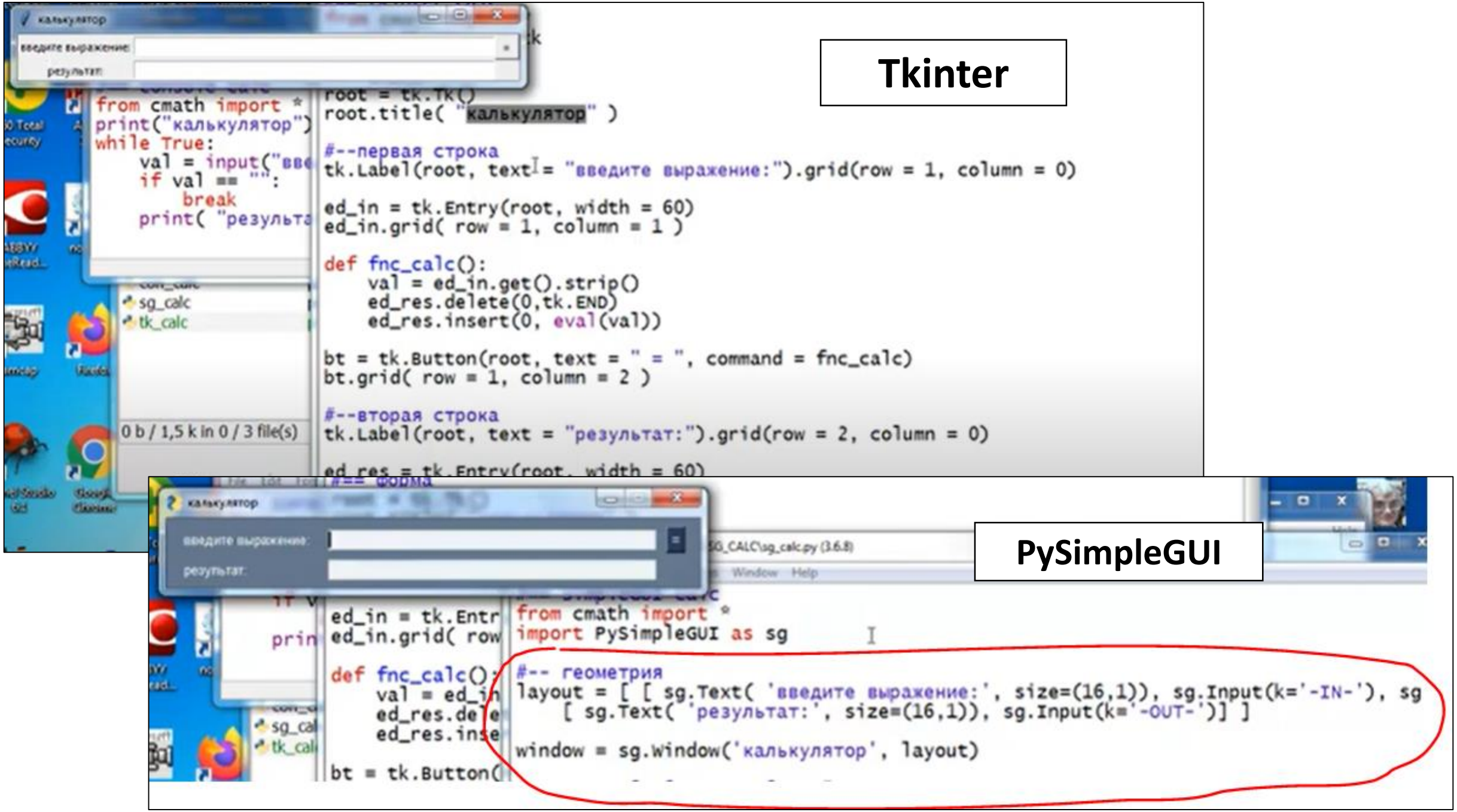
✓ Поддержка нескольких бэкендов (Tkinter, Qt, wxPython).

Недостатки:

✓ Ограниченная гибкость по сравнению с PyQt или wxPython.

Документация:
https://pysimplegui.readthedocs.io/




калькулятор
введите выражение:
результат:
from cmath import *
print("калькулятор")
while True:
val = input("вве
if val == "":
break
print("результа
sg_calc
tk_calc
0 b / 1,5 k in 0 / 3 file(s)
root = tk.Tk()
root.title("калькулятор")
#--первая строка
tk.Label(root, text= "введите выражение:").grid(row = 1, column = 0)
ed_in = tk.Entry(root, width = 60)
ed_in.grid(row = 1, column = 1)
def fnc_calc():
val = ed_in.get().strip()
ed_res.delete(0,tk.END)
ed_res.insert(0, eval(val))
bt = tk.Button(root, text = " = ", command = fnc_calc)
bt.grid(row = 1, column = 2)
#--вторая строка
tk.Label(root, text = "результат:").grid(row = 2, column = 0)
ed_res = tk.Entry(root, width = 60)
Tkinter
калькулятор
введите выражение:
результат:
from cmath import *
import PySimpleGUI as sg
#-- геометрия
layout = [[sg.Text('введите выражение:', size=(16,1)), sg.Input(k='-IN-'), sg
[sg.Text('результат:', size=(16,1)), sg.Input(k='-OUT-')]]
window = sg.Window('калькулятор', layout)
PySimpleGUI

5. Textual

Библиотека для создания текстовых пользовательских интерфейсов (TUI) в терминале.

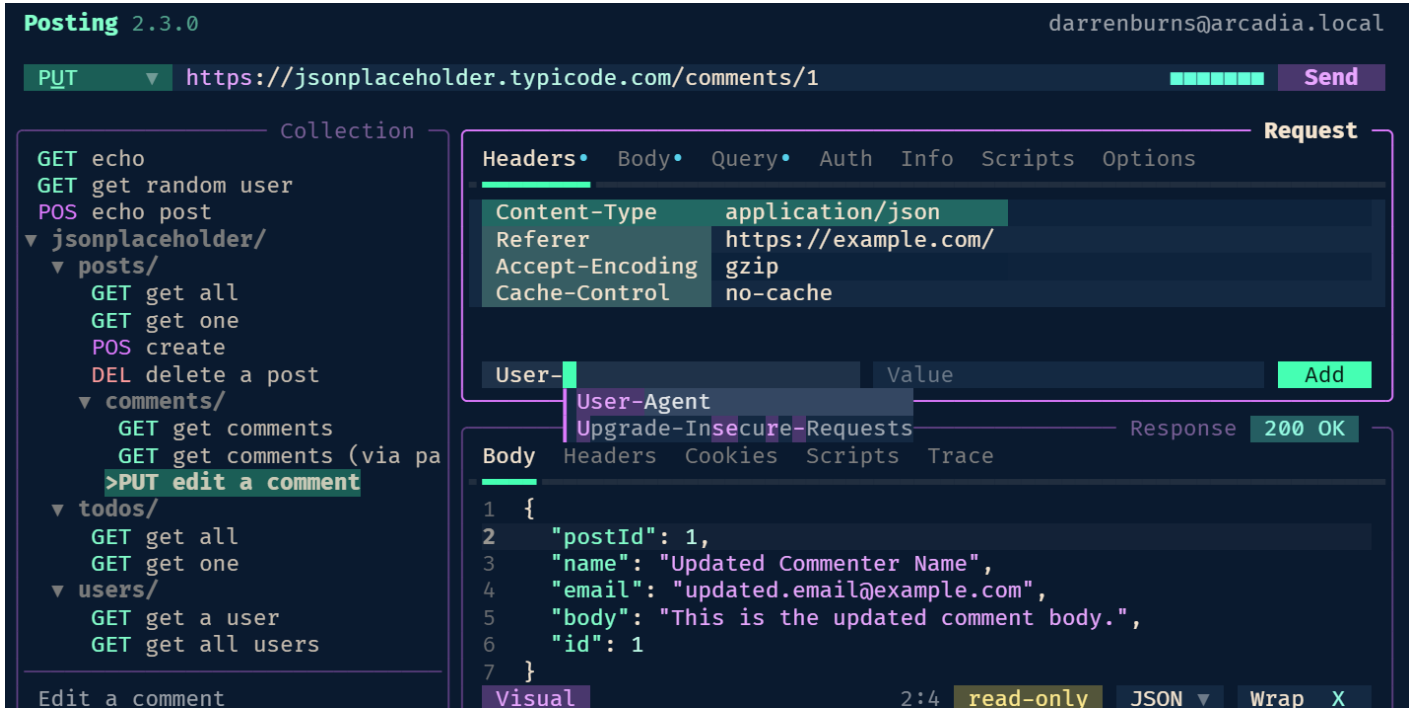
Преимущества:

✓ Подходит для создания CLI-приложений с улучшенным интерфейсом.

Недостатки:

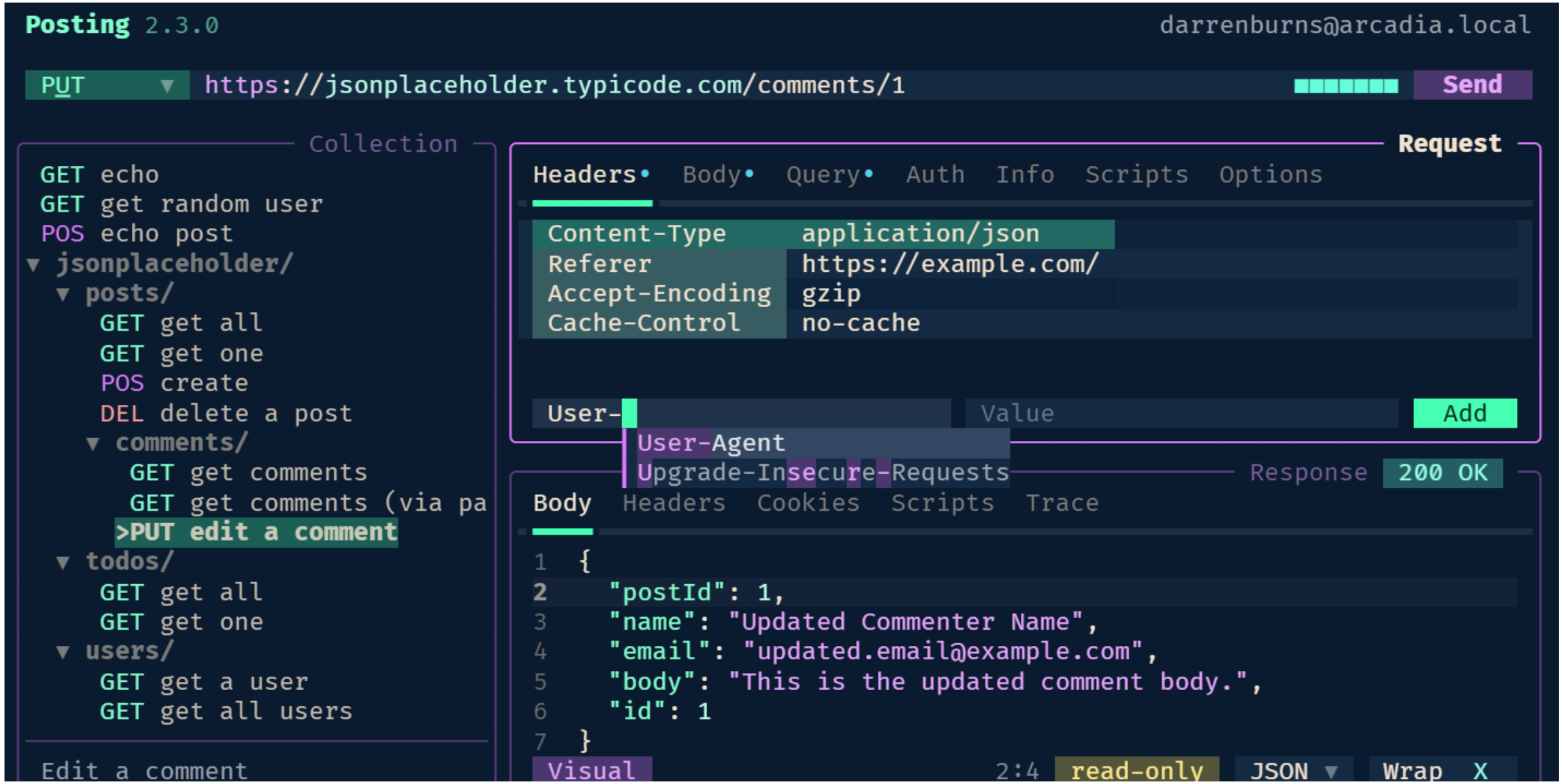
✓ Ограничена текстовым интерфейсом.

Документация:
https://textual.textualize.io/



Пример пользовательского интерфейса в терминале на Textual

Пример пользовательского интерфейса в терминале на Textual



Работа с Tkinter

Работу с Tkinter можно представить в виде четырёх шагов::

- ✓ Подключаем библиотеку Tkinter с помощью директивы import.
- ✓ Создаём главное окно приложения.
- ✓ Добавляем виджеты — визуальные элементы.
- ✓ Создаём главный цикл событий — он включает в себя все события, происходящие при взаимодействии пользователя с интерфейсом.

1
2
3
4

```python
import tkinter as tk

# Создаем главное окно
root = tk.Tk()
root.title("Форма с кнопкой")  # Заголовок окна

# Создаем кнопку
button = tk.Button(root, text="Нажми меня")  # Кнопка с текстом "Нажми меня"
button.pack(pady=20)  # Размещаем кнопку на форме с отступом

# Запускаем главный цикл обработки событий
root.mainloop()
```

Работа с Tkinter

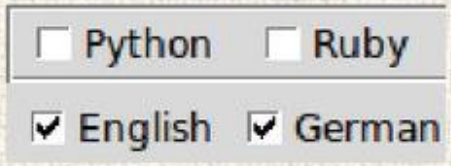
Ключевые объекты в работе с Tkinter — виджеты. Это аналоги тегов из HTML, которые позволяют создавать интерактивные и неинтерактивные элементы, например надписи или кнопки.

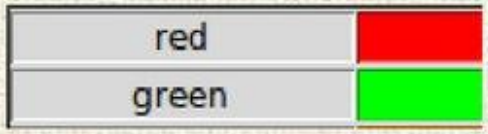
Всего их 18, чаще всего используют следующие:

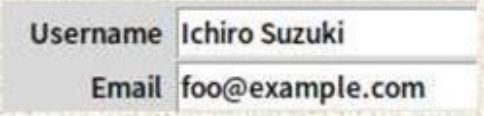
- Button — кнопки;
- Canvas — «холст», на котором рисуют графические фигуры;
- Entry — виджет для создания полей ввода;
- Label — контейнер для размещения текста или изображения;
- Menu — виджет для создания пунктов меню.

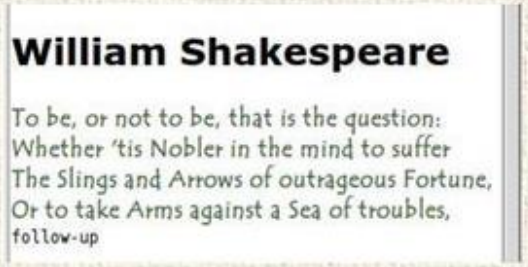
Типы виджетов

- Button (Кнопка). Простая кнопка для вызова некоторых действий (выполнения определенной команды).
- Checkbutton (Флажок). Кнопка, которая умеет переключаться между двумя состояниями при нажатии на нее.
- Label (Надпись). Может показывать текст или графическое изображение.
- Entry (Однострочное текстовое поле ввода). Горизонтальное поле, в которое можно ввести строку текста.
- Text (Многострочное текстовое поле). Позволяет редактировать и форматировать текст с использованием различных стилей, внедрять в текст рисунки и даже окна.



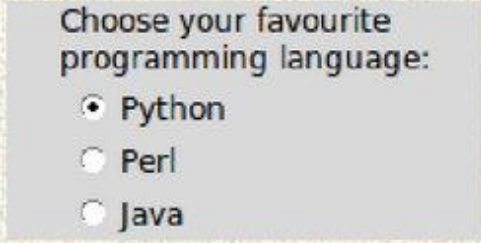


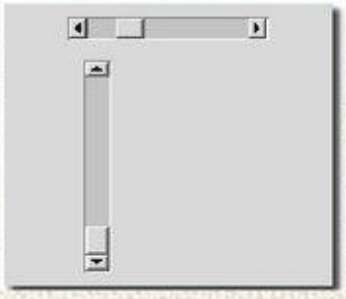


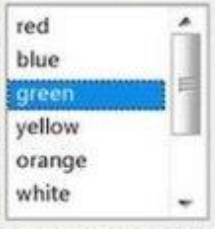


Типы виджетов

- Radiobutton (Селекторная кнопка). Кнопка для представления одного из альтернативных значений.
- Scrollbar (Полоса прокрутки). Служит для отображения величины прокрутки (вертикальной, горизонтальной) в других виджетах.
- Scale (Шкала). Служит для задания числового значения путем перемещения «бегунка» в определенном диапазоне.
- Listbox (Список). Прямоугольная рамка со списком, из которого пользователь может выделить один или несколько элементов.
- Canvas (Рисунок). Основа для вывода графических примитивов.

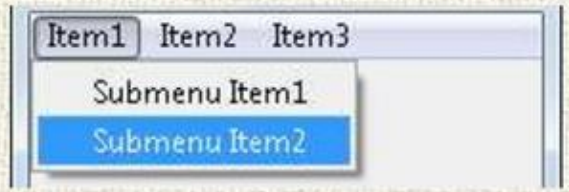


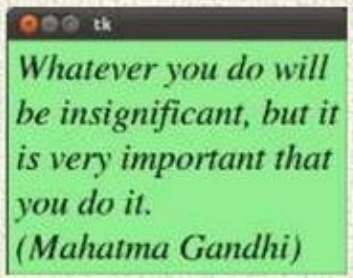


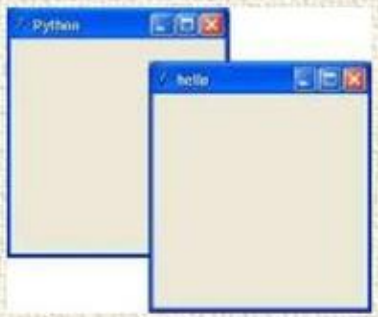


Типы виджетов

- Frame (Рамка). Виджет, который содержит в себе другие визуальные компоненты.
- Menu (Меню). Элемент, с помощью которого можно создавать всплывающие (popup) и ниспадающие (pulldown) меню.
- Menubutton (Кнопка-меню). Кнопка с ниспадающим меню.
- Message (Сообщение). Аналогично надписи, но позволяет переносить длинные строки и менять размер по требованию менеджера расположения.
- Toplevel (Окно верхнего уровня). Показывается как отдельное окно и содержит внутри другие виджеты.







```python
import tkinter as tk
from tkinter import messagebox

# Функция, которая вызывается при нажатии на кнопку
def show_message():
    messagebox.showinfo("Сообщение", "Привет, мир!")  # Выводим сообщение

# Создаем главное окно
root = tk.Tk()
root.title("Пример с кнопкой")  # Заголовок окна

# Создаем кнопку
button = tk.Button(root, text="Нажми меня", command=show_message)
button.pack(pady=20)  # Размещаем кнопку на форме с отступом

# Запускаем главный цикл обработки событий
root.mainloop()
```

Позиционирование элементов в окне

Для размещения виджетов на форме в Tkinter есть несколько методов (упаковщиков):

- ✓ **pack()** — используется, когда мы работаем с контейнерами для элементов. Позволяет позиционировать кнопки, надписи или другие элементы внутри контейнеров.
- ✓ **place()** — позволяет позиционировать элементы, указывая точные координаты.
- ✓ **grid()** — размещает элементы по ячейкам условной сетки, разделяющей окно приложения.

Метод **pack ()** выполняет простое расположение без привязки к месту.

```
w = Label(root, text="Red", bg="red", fg="white")
w.pack()
w = Label(root, text="Green", bg="green", fg="black")
w.pack()
w = Label(root, text="Blue", bg="blue", fg="white")
w.pack()
```


```
# ----------------------------------------------------------------------

w = Label(root, text="Red", bg="red", fg="white")
w.pack(fill=X)
w = Label(root, text="Green", bg="green", fg="black")
w.pack(fill=X)
w = Label(root, text="Blue", bg="blue", fg="white")
w.pack(fill=X)
```


```
# ----------------------------------------------------------------------

w = Label(root, text="Red", bg="red", fg="white")
w.pack(side=LEFT)
w = Label(root, text="Green", bg="green", fg="black")
w.pack(side=LEFT)
w = Label(root, text="Blue", bg="blue", fg="white")
w.pack(side=LEFT)
```

Метод **place()** позволяет точно указать координаты и размеры виджетов.

```
import tkinter as tk

# Создаем главное окно
root = tk.Tk()
root.title("Пример размещения с place()")  # Заголовок окна
root.geometry("300x200")  # Устанавливаем размер окна

# Создаем поле ввода и размещаем его с помощью place()
entry = tk.Entry(root, width=20)
entry.place(x=50, y=60)  # Размещаем поле ввода на координатах (50, 60)

# Создаем кнопку и размещаем ее с помощью place()
button = tk.Button(root, text="Нажми меня")
button.place(x=50, y=100, width=100, height=30)  # Размещаем кнопку на координатах (50, 100) с у
казанием ширины и высоты

# Запускаем главный цикл обработки событий
root.mainloop()
```

Метод grid() позволяет организовать виджеты в виде таблицы (сетки)

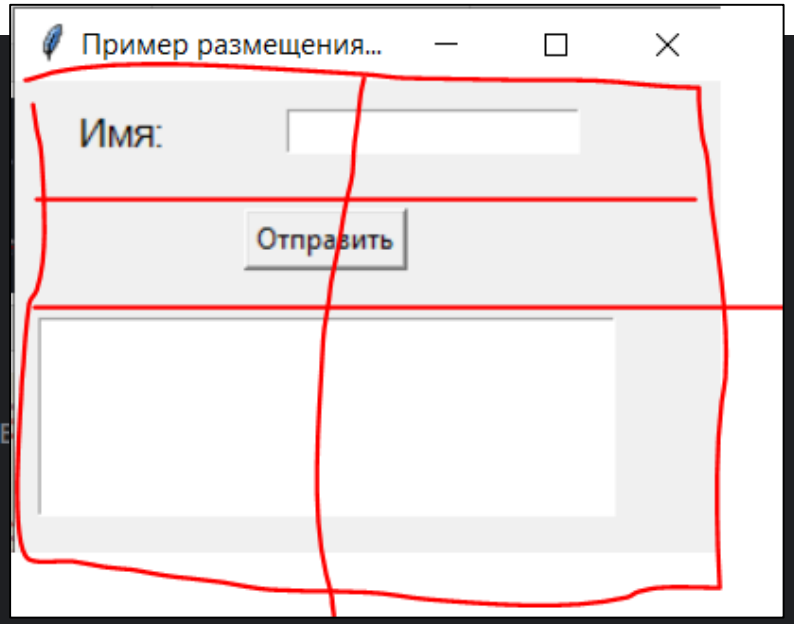
```
root = tk.Tk()
root.title("Пример grid()")  # Заголовок окна
root.geometry("300x200")  # Устанавливаем размер окна

# Создаем метку и размещаем ее с помощью grid()
label = tk.Label(root, text="Имя:", font=("Arial", 12))
label.grid(row=0, column=0, padx=10, pady=10)  # Размещаем метку в

# Создаем поле ввода и размещаем его с помощью grid()
entry = tk.Entry(root, width=20)
entry.grid(row=0, column=1, padx=10, pady=10)  # Размещаем поле ввода в первой строке и втором столбце

# Создаем кнопку и размещаем ее с помощью grid()
button = tk.Button(root, text="Отправить")
button.grid(row=1, column=0, columnspan=2, pady=10)  # Размещаем кнопку во второй строке, объединяя два

# Создаем текстовое поле для вывода и размещаем его с помощью grid()
output = tk.Text(root, height=5, width=30)
output.grid(row=2, column=0, columnspan=2, padx=10, pady=10)  # Размещаем текстовое поле в третьей строк
```



Преимущества метода `grid()`:

- Удобство для создания структурированных интерфейсов.
- Легкость управления расположением виджетов в виде таблицы.
- Поддержка объединения ячеек (`columnspan`, `rowspan`).

Дополнительные параметры `grid()`:

- `sticky`: Указывает, как виджет должен "растягиваться" внутри ячейки. Например:
  - `sticky="n"` — прижимает к верхнему краю.
  - `sticky="we"` — растягивает по горизонтали.
  - `sticky="nsew"` — растягивает во все стороны.
- `rowspan`: Объединяет несколько строк.
- `columnspan`: Объединяет несколько столбцов.
- `ipadx`, `ipady`: Внутренние отступы внутри виджета.
- `padx`, `pady`: Внешние отступы вокруг виджета.

Пример: кнопка растягивается по горизонтали
button.grid(row=1, column=0, columnspan=2, **sticky="we**")

Параметры виджетов

```
from tkinter import *

root = Tk()

button = Button(root,
                text="Это кнопка",  # надпись на кнопке
                width=30,  # ширина
                height=5,  # высота
                bg="white",  # цвет фона и надписи
                fg="blue",
                font="Verdana 12",  # шрифт текста
                activebackground="black",  # цвет фона и надписи
                activeforeground="red")  # при наведении курсора

button.pack()
root.mainloop()
```

Работа с изображениями

Существует много библиотек для работы с изображениями, мы рассмотрим две:

- ✓ **Pillow (PIL)** — это расширение библиотеки Python Imaging Library (PIL). Она предоставляет функции для работы с изображениями, такие как изменение размера, поворот, наложение фильтров и многое другое.
- ✓ **OpenCV** — это открытая библиотека компьютерного зрения, которая предоставляет функции для работы с изображениями и видео. **Она широко используется для обнаружения объектов, распознавания лиц, анализа движения и многих других задач компьютерного зрения**.

Pillow

Pillow — это свободно распространяемая библиотека для работы с изображениями на Python с открытым исходным кодом, которая добавляет вашему коду поддержку открытия, изменения и сохранения изображений в различных расширениях.

Установка :

```
pip install pillow
```

Самый важный класс в Pillow — это **класс Image**, определенный в одноименном модуле.

```
from PIL import Image
```

Загрузка изображения и изменение размера изображения с использованием Pillow

```python
from PIL import Image

image = Image.open("example.jpg")
new_size = (100, 100)

# Изменение размера изображения
resized_image = image.resize(new_size)
resized_image.save("resized_example.jpg")
```

```
from PIL import Image
sample = Image.open('pena.jpg')

sample.show() #Открывает в новом окн

sample.format
'JPEG'
sample.size
(640, 640)
sample.mode
'RGB'
```

Можно посмотреть свойства изображения, например:

- формат
- размер
- цветовой режим

```python
from PIL import Image

# Загрузка изображения
image = Image.open("example.jpg")

# Сохранение изображения в другом формате
image.save("example.png")
```

```python
from PIL import Image

image = Image.open("example.jpg")
new_size = (100, 100)

# Изменение размера изображения
resized_image = image.resize(new_size)
resized_image.save("resized_example.jpg")
```

Фильтрация изображений

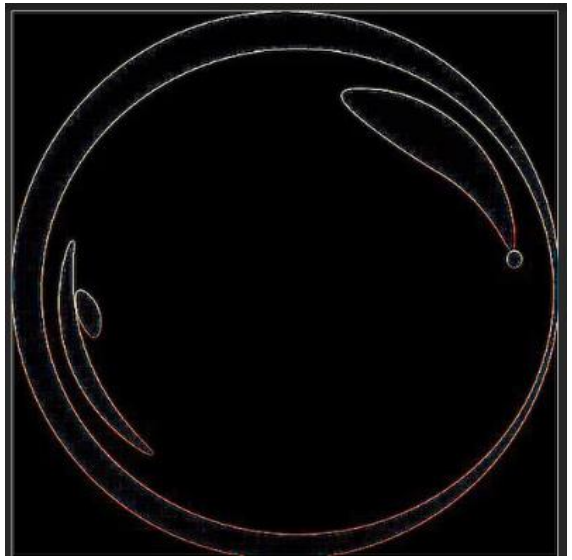
Фильтрация — это метод изменения или улучшения изображения. Например, вы можете отфильтровать изображение, чтобы выделить определенные особенности или удалить другие.

Фильтрация изображений используется для получения различных результатов, как, например,- сглаживание, повышение резкости, удаление шума и обнаружение краев.

В библиотеке Pillow доступно множество фильтров, включая BLUR, BoxBlur, CONTOUR, FIND_EDGES, Filter, GaussianBlur, Kernel, MaxFilter, MedianFilter, SHARPEN, SMOOTH и т.д.

Найдем края на изображении ниже, используя фильтр FIND_EDGES

```
from PIL import Image
from PIL import Image, ImageFilter
image = Image.open('pena.jpg')
edges = image.filter(ImageFilter.FIND_EDGES)
edges.show()
```



Для улучшения изображения можно использовать классы в **ImageEnhance** модуле.

Таким образом можно настроить контрастность, яркость, цветовой баланс и резкость.

```
from PIL import ImageEnhance


enh = ImageEnhance.Contrast(im)
enh.enhance(1.3).show("30% more contrast")
```

Справка по этому модулю

Пример приложения с библиотеками Tkinter и Pillow

На форме расположена кнопка, которая открывает диалоговое окно выбора файла с изображением и выводит изображение в label на форму

Этот код можно скопировать ↓

```
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk, ImageFilter

# Функция для загрузки и отображения изображения
def load_image():
    global img, img_label, img_tk
    # Открываем диалоговое окно для выбора файла
    file_path = filedialog.askopenfilename(filetypes=[("Image files",
"*.jpg;*.jpeg;*.png;*.bmp;*.gif")])

    if file_path:
        img = Image.open(file_path)
        img.thumbnail((400, 400))  # Устанавливаем максимальный размер 400x400
        img_tk = ImageTk.PhotoImage(img) # Преобразуем изображение для Tkinter
        img_label.config(image=img_tk) # Создаем метку для отображения изображения
```

Продолжение >>>

Этот код можно скопировать ↓

```
# Создаем основное окно
root = tk.Tk()
root.title("Обработка изображения")
root.geometry("600x600+400+200")

# Создание кнопки
load_button = tk.Button(root, text="Load Image", command=load_image)
load_button.pack(pady=5)

# Метка для отображения изображения
img_label = tk.Label(root)
img_label.pack()

root.mainloop()
```

```
# Функция для загрузки и отображения изображения
def load_image():
    global img, img_label, img_tk
```

Ключевое слово *global* используется для объявления переменных как глобальных. Это означает, что переменные, объявленные как *global*, будут доступны и изменяемы во всей области видимости программы, а не только внутри функции, где они были изменены.

Этот код можно скопировать ↓

```
def rotate_image():
    global img, img_label, img_tk
    if img:
        img = img.rotate(-90, expand=True)  # Поворот на 90 градусов по часовой стрелке
        img_tk = ImageTk.PhotoImage(img)
        img_label.config(image=img_tk)
```

В нашем примере изменения переменных, содержащих изображения, будут доступны всем функциям, например функции Поворота.

Когда использовать Pillow?

Pillow проще в использовании для базовых задач, таких как:

- ✓ Открытие, редактирование и сохранение изображений.
- ✓ Простые операции (изменение размера, поворот, обрезка, простые преобразования).
- ✓ Добавление текста или графики на изображение.

OpenCV

OpenCV — это open source библиотека компьютерного зрения, которая предназначена для анализа, классификации и обработки изображений. Широко используется в таких языках как C, C++, Python и Java.

Пример загрузки изображения при помощи OpenCV

Этот код можно скопировать ↓

```
import tkinter as tk
from tkinter import filedialog
import cv2
import numpy as np

# Функция для загрузки и вывода изображения
def load_image():
            # Открываем диалоговое окно для выбора файла
    file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.jpg;*.jpeg;*.png;*.bmp;*.gif")])
    if file_path:  # Если файл не выбран, выходим

            # Загружаем изображение с помощью OpenCV
        image = cv2.imread(file_path)
        image = cv2.resize(image, (400, 400), cv2.INTER_NEAREST)
            # Преобразуем изображение в формат, который может отображать Tkinter
        image = np.ascontiguousarray(image)  # Убедимся, что массив непрерывный
        photo = tk.PhotoImage(data=cv2.imencode('.png', image)[1].tobytes())

            # Обновляем метку для отображения изображения
        image_label.config(image=photo)
            # Сохраняем ссылку, чтобы изображение не удалялось сборщиком мусора
        image_label.image = photo
```

основные преимущества OpenCV перед Pillow

1. Широкий спектр функций для компьютерного зрения

- OpenCV предоставляет множество инструментов для задач компьютерного зрения, таких как:
  - Детекция объектов (например, лиц, глаз, машин).
  - Распознавание образов (использование нейронных сетей, SVM и других методов).
  - Сегментация изображений.
  - Обработка видео (трекинг объектов, анализ движения).
- Pillow ориентирована в основном на базовую обработку изображений и не поддерживает такие функции.

2. Поддержка работы с видео

- OpenCV позволяет работать с видеофайлами и видеопотоками:
  - Захват видео с камеры.
  - Обработка кадров в реальном времени.
  - Сохранение видео.
- Pillow не поддерживает работу с видео, так как это библиотека для статичных изображений.

3. Высокая производительность

- OpenCV оптимизирована для высокопроизводительных операций:
  - Реализация на C++ с Python-интерфейсом.
  - Поддержка многопоточности и GPU (через модуль cv2.cuda).
  - Эффективная обработка больших изображений и видео.
- Pillow, хотя и быстрая, не предназначена для таких высоконагруженных задач.

4. Расширенные методы обработки изображений

- OpenCV предоставляет больше возможностей для сложной обработки изображений:
  - Морфологические операции (эрозия, дилатация, открытие, закрытие).
  - Фильтры (Гаусса, Собеля, Лапласа и др.).
  - Поиск контуров и работа с ними.
  - Коррекция освещения и цвета.
- Pillow ограничена базовыми операциями, такими как изменение размера, поворот и применение простых фильтров.

5. Интеграция с машинным обучением

- OpenCV интегрирована с популярными библиотеками машинного обучения, такими как TensorFlow, PyTorch и Keras.
- Поддержка предобученных моделей для детекции объектов, классификации изображений и других задач.
- Pillow не предоставляет таких возможностей.

6. Работа с 3D-изображениями и калибровка камер

- OpenCV поддерживает:
  - Калибровку камер (устранение искажений).
  - Работу с 3D-изображениями и стереозрением.
  - Восстановление 3D-сцен.
- Pillow не имеет таких функций.

7. Поддержка различных платформ

- OpenCV поддерживает:
  - Windows, Linux, macOS.
  - Мобильные платформы (Android, iOS).
  - Встраиваемые системы (например, Raspberry Pi).
- Pillow также кроссплатформенна, но OpenCV лучше адаптирована для работы на устройствах с ограниченными ресурсами.

8. Большое сообщество и документация

- OpenCV имеет огромное сообщество и множество примеров, tutorials и документации.
- Поддержка множества языков программирования (C++, Python, Java и др.).
- Pillow также имеет хорошую документацию, но OpenCV более популярна в профессиональной среде.

9. Поддержка GPU и аппаратного ускорения

- OpenCV поддерживает использование GPU для ускорения операций (через модуль cv2.cuda).
- Это особенно полезно для задач, требующих высокой производительности, таких как обработка видео или работа с нейронными сетями.
- Pillow не поддерживает GPU.

10. Гибкость и расширяемость

- OpenCV позволяет легко интегрироваться с другими библиотеками, такими как NumPy, SciPy, Matplotlib и др.
- Поддержка пользовательских алгоритмов и расширений.

OpenCV — это мощный инструмент для профессиональной обработки изображений и компьютерного зрения, тогда как Pillow лучше подходит для простых задач.

Если ваша задача связана с анализом изображений, видео или машинным обучением, OpenCV будет предпочтительным выбором.