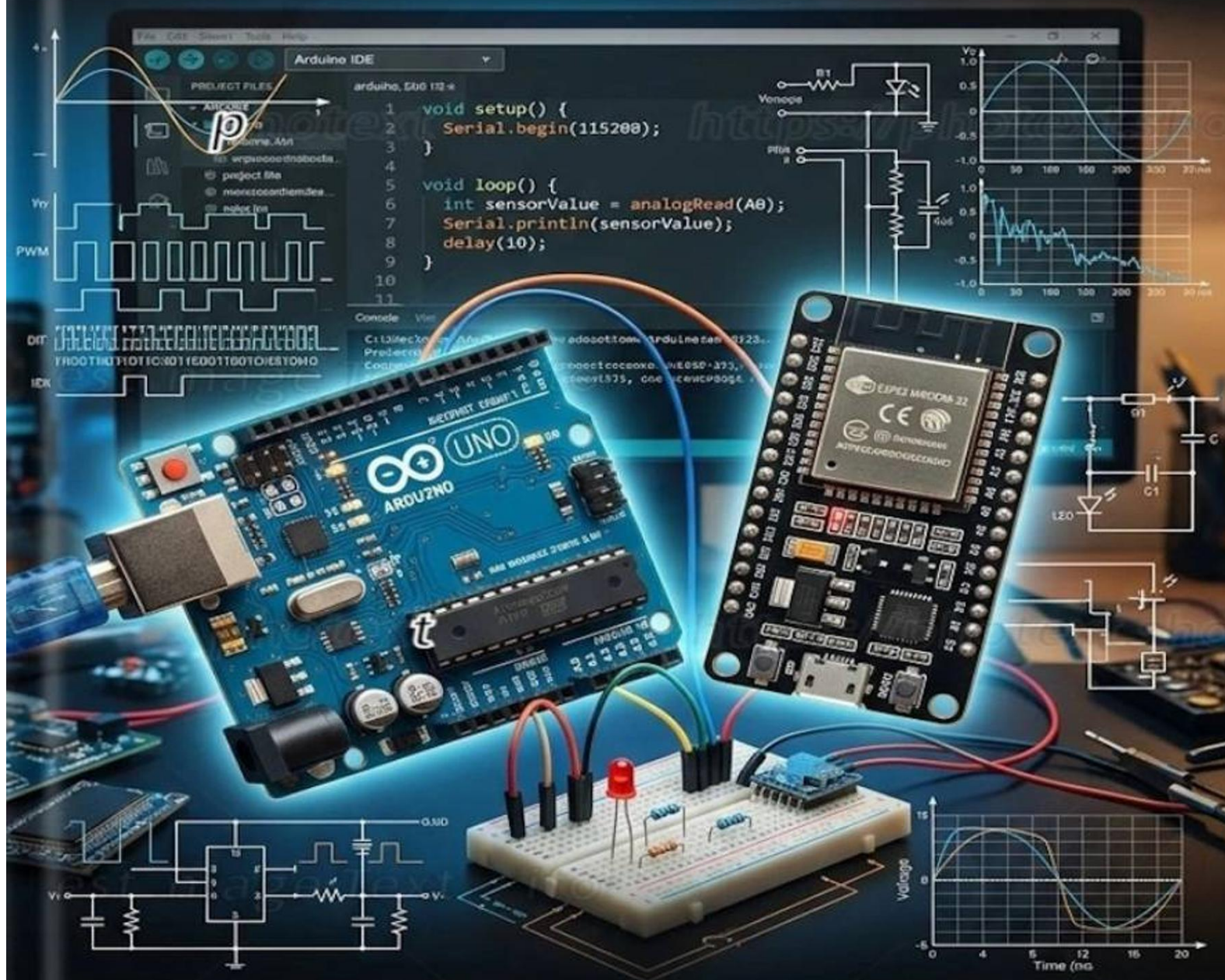


# ОТ ДАТЧИКА ДО ИНТЕРФЕЙСА

Практическое руководство по созданию инженерных десктоп приложений на Python и web-технологиях



**АВТОР: НАВИ ТУЛАГ**

Проекты на Arduino и ESP32.  
Код, схемы, инструкции

Нави Тулаг

**От датчика до интерфейса.  
Практическое руководство  
по созданию инженерных  
десктоп приложений на  
Python и web-технологиях**

«Автор»

2026

## **Тулаг Н.**

От датчика до интерфейса. Практическое руководство по созданию инженерных десктоп приложений на Python и web-технологиях / Н. Тулаг — «Автор», 2026

Эта книга — практическое руководство для инженеров, исследователей и студентов, желающих создать собственные десктоп-приложения для работы с оборудованием. Вы научитесь соединять физический мир датчиков и микроконтроллеров (Arduino, ESP32) с цифровым миром современных интерфейсов на Python и веб-технологиях. Уникальная особенность книги — режим эмуляции, позволяющий тестировать приложения без подключенного железа. Пошаговые примеры ведут от простого калькулятора до полноценной системы удалённого мониторинга. Вы узнаете, как упаковать проект в .exe для коллег, обеспечить надёжность связи и выбрать правильный фреймворк (Eel, Electron, Neutralino) под задачу. Мы сохранили стиль первой книги: от философии к практике, с готовыми решениями и объяснением каждого шага. Это не просто код, это инструмент для автоматизации вашей инженерной работы. Превратите свои скрипты в профессиональные продукты и освободите время для творчества.

© Тулаг Н., 2026

© Автор, 2026

# Содержание

|                                                                   |    |
|-------------------------------------------------------------------|----|
| Предисловие. Между физическим и цифровым миром                    | 7  |
| Часть 1. Введение в инженерную разработку ПО                      | 10 |
| Зачем инженеру своё приложение?                                   | 10 |
| 1.1. Excel и консоль — это не панацея                             | 10 |
| 1.2. Причина №1: Доступ к оборудованию                            | 10 |
| 1.3. Причина №2: Визуализация в реальном времени                  | 11 |
| 1.4. Причина №3: Автономная работа                                | 11 |
| 1.5. Причина №4: Производительность и автоматизация               | 11 |
| 1.6. Когда своё приложение — избыточно?                           | 11 |
| 1.7. Что мы узнали?                                               | 11 |
| Обзор архитектур: традиционные GUI (Tkinter, PyQt) vs WEB-обёртки | 13 |
| 2.1. Классический путь: Tkinter и PyQt                            | 13 |
| 2.2. Современный путь: Веб-обёртки (Hybrid Apps)                  | 14 |
| 2.3. Сравнительная таблица: Что выбрать инженеру?                 | 15 |
| 2.4. Почему мы выбираем Eel для этой книги?                       | 15 |
| 2.5. Когда всё-таки стоит выбрать Tkinter?                        | 15 |
| Знакомство с инструментами: Python Eel, Electron, Neutralino.     | 17 |
| Их философия и применение в инженерии                             |    |
| 3.1. Python Eel: «Мой скрипт заслуживает интерфейс»               | 17 |
| 3.2. Electron: «Веб-приложение как десктопный продукт»            | 19 |
| 3.3. Neutralino: «Лёгкий и современный»                           | 20 |
| 3.4. Сравнение в одном взгляде                                    | 21 |
| 3.5. Когда что выбирать?                                          | 22 |
| 3.6. Наша философия для этой книги                                | 23 |
| 3.7. Реальный пример: От Arduino до облака                        | 23 |
| Критерии выбора: когда использовать какой фреймворк?              | 25 |
| 4.1. Пять вопросов перед выбором                                  | 25 |
| 4.2. Сравнительная таблица: объективные метрики                   | 26 |
| 4.3. Алгоритм выбора: пошаговая инструкция                        | 26 |
| 4.4. Когда НИ ОДИН из трёх не подходит?                           | 27 |
| 4.5. Частые ловушки новичков                                      | 27 |
| 4.6. Наш выбор для этой книги                                     | 28 |
| Часть 2. Python Eel — основной инструмент инженера                | 30 |
| Быстрый старт с Eel. Установка, структура проекта, первый запуск  | 30 |
| 5.1. Почему Eel? Философия быстрого старта                        | 30 |
| 5.2. Установка Eel: Пошагово                                      | 30 |
| 5.3. Структура проекта: Правильно с первого дня                   | 32 |
| 5.4. Первое приложение: «Калькулятор инженера»                    | 32 |
| 5.5. Запуск приложения                                            | 38 |
| 5.6. Как это работает? Архитектура под капотом                    | 39 |
| 5.7. Режимы запуска Eel                                           | 39 |
| 5.8. Частые ошибки и решения                                      | 40 |
| 5.9. Подготовка к следующему шагу: Режим эмуляции                 | 41 |

|                                                                                     |    |
|-------------------------------------------------------------------------------------|----|
| Связка Python и JS. Вызов функций, передача данных, асинхронность без блокировки UI | 43 |
| 6.1. Двусторонний мост: как работает связь                                          | 43 |
| 6.2. Вызов Python из JavaScript: правильный синтаксис                               | 44 |
| 6.3. Передача сложных данных                                                        | 45 |
| 6.4. Вызов JavaScript из Python                                                     | 47 |
| 6.5. Асинхронность: как не заблокировать UI                                         | 48 |
| 6.6. Режим эмуляции: тестирование без железа                                        | 51 |
| 6.7. Обработка ошибок на обеих сторонах                                             | 52 |
| 6.8. Частые ошибки и решения                                                        | 54 |
| Конец ознакомительного фрагмента.                                                   | 56 |

**Нави Тулаг**  
**От датчика до интерфейса.**  
**Практическое руководство по**  
**созданию инженерных десктоп**  
**приложений на Python и web-технологиях**  
**ОТ ДАТЧИКА ДО ИНТЕРФЕЙСА**

Практическое руководство по созданию инженерных десктоп-приложений на Python и web-технологиях

**Нави Тулаг**

## Предисловие. Между физическим и цифровым миром

Представьте себе на мгновение современную инженерную лабораторию.

С одной стороны — физический мир. Датчики температуры, вибрации, давления. Микроконтроллеры Arduino и ESP32, которые снимают показания 24/7. COM-порты, по которым бегут байты данных. Провода, паяльник, осциллограф. Это мир сигналов, шумов и реальных измерений.

С другой стороны — цифровой мир. Графики в реальном времени, базы данных, облачные сервисы, веб-интерфейсы. Это мир визуализации, анализа и принятия решений.

Долгое время эти миры были разделены.

Инженеры, умеющие работать с железом, часто останавливались на консоли или Excel. Веб-разработчики создавали красивые интерфейсы, но не могли получить данные с датчика. Python-скрипты анализировали данные, но не имели удобного GUI для коллег.

Но что, если бы существовал мост?

Мост, который позволяет принять данные с Arduino через COM-порт и сразу отобразить их на красивом графике в окне приложения? Мост, который позволяет превратить скрипт на Python в полноценную программу с кнопками и меню, которую можно отправить коллеге одним файлом? Мост, который позволяет получать данные с ESP32 из любой точки мира через облако?

Этот мост был построен. И он имеет несколько мощных опор: Python Eel, Electron и Neutralino.

Эта книга — ваш путеводитель по этому мосту.

Почему эта книга? И для кого она?

Когда я только начинал создавать инструменты для своей лаборатории, я столкнулся с парадоксом. Информации было много, но она была разрозненна. Тutorials по веб-разработке не говорили о COM-портах. Примеры для Arduino заканчивались на «мигающем светодиоде» и простом взаимодействии с датчиками. Мне приходилось методом проб и ошибок, тратя много времени, собирать по крупицам понимание: как сделать так, чтобы кнопка в интерфейсе реально включала реле?

Я написал эту книгу, чтобы сэкономить вам это время и избавить от этих ошибок.

Эта книга для Вас:

| Если вы...                        | Эта книга даст вам...                                                                  |
|-----------------------------------|----------------------------------------------------------------------------------------|
| Инженер-экспериментатор           | Быстрый способ создать панель управления для вашей установки без изучения сложного GUI |
| Студент технической специальности | Дипломный проект с реальным железом и современным интерфейсом                          |
| Исследователь/лаборант            | Инструмент для сбора данных без зависимости от IT-отдела                               |
| Разработчик на Python             | Возможность «прикрутить» удобный фронтенд к своим скриптам за вечер                    |

*Таблица 1. Эта книга для Вас*

Наша философия: от простого к сложному

В интернете полно священных войн на холиварную тему «Какой фреймворк лучше?». Эта книга не участвует в них. Наша главная цель — дать вам инструмент для решения задачи.

Мы пойдём по пути постепенного усложнения:

Часть 1: Поймём, зачем вообще нужно своё приложение.

Часть 2: Освоим Python Eel — самый простой способ для инженера.

Часть 3: Подключим Arduino и научимся работать с COM-портами.

Часть 4: Выйдем в IoT — получим данные с ESP32 из облака через MQTT.

Часть 5: Рассмотрим альтернативы и научимся упаковывать приложения для коллег.

Уникальная особенность: Режим эмуляции

Одна из главных проблем инженерного ПО — зависимость от железа. Что делать, если Arduino нет под рукой? Или она в другой лаборатории?

В этой книге вы узнаете, как внедрить режим эмуляции (DEBUG\_MODE). Вы сможете тестировать интерфейс и логику приложения, генерируя правдоподобные данные программно. Это ускорит разработку в 3–5 раз и позволит демонстрировать приложение заказчику без необходимости везти с собой установку.

Что вас ждёт внутри?

Каждая глава — это готовое решение, которое можно скопировать и адаптировать:

Режим эмуляции — тестируйте код без Arduino/ESP32

Графики в реальном времени — Chart.js + Python

Работа с COM-портами — pyserial, обработка ошибок

IoT-мониторинг — MQTT, HiveMQ Cloud, GitHub Pages

Упаковка в .exe — PyInstaller, один файл для коллег

Исходный код всех приложений, упомянутых в статье, доступен в открытом репозитории на GitHub:

[https://github.com/Book-From-the-sensor-to-the-interface/sketch\\_for\\_book.git](https://github.com/Book-From-the-sensor-to-the-interface/sketch_for_book.git)

Технологии — это не просто инструменты. Это расширители возможностей. Они стирают границы между «железом» и «программой». Моя задача как автора — не просто научить

вас пользоваться тремя фреймворками, а дать вам свободу создавать инструменты, которые облегчают вашу инженерную работу.

Готовы стереть границы между датчиком и интерфейсом? Тогда начнём наше путешествие.

*С наилучшими пожеланиями,  
Нави Тулаг.*

## Часть 1. Введение в инженерную разработку ПО

### Зачем инженеру своё приложение?

О чём эта глава?

Вы, вероятно, слышали фразу: «*Зачем писать приложение, если есть Excel или Serial Monitor?*»

Иногда это звучит как истина. Но на практике — готовые инструменты часто не подходят под специфические задачи лаборатории или производства.

В этой главе мы разберём четыре ключевые причины, по которым инженеру стоит писать своё ПО:

- Производительность (автоматизация рутины)

- Доступ к оборудованию (COM-порты, GPIO, датчики)

- Автономная работа (без интернета, в цеху, в «поле»)

- Визуализация и контроль (графики в реальном времени, кнопки управления)

Вы поймёте, почему такие задачи, как «сбор данных с термопары» или «контроль вибрации двигателя», лучше решать своим приложением, чем универсальными инструментами.

#### 1.1. Excel и консоль — это не панацея

Представьте: вы проводите серию экспериментов. Вам нужно снимать показания с датчика каждые 100 мс, сохранять их в файл и сразу видеть график, чтобы понять, не вышел ли параметр за допустимые пределы.

В Excel:

- Нет прямого доступа к COM-порту (нужны макросы или плагины).

- Обновление данных в реальном времени тормозит на больших объёмах.

- Нельзя отправить команду «Включить нагрев» обратно на устройство.

В Serial Monitor (Arduino IDE):

- Только текст, нет графиков.

- Нет сохранения данных в удобном формате (CSV/JSON).

- Нельзя создать кнопки управления.

А в своём приложении — всё это возможно. И даже необходимо.

#### 1.2. Причина №1: Доступ к оборудованию

Веб-приложения (в браузере) работают в «песочнице». Они не знают, где на диске лежит файл, не видят USB-устройства (без сложных API).

Инженерное десктоп-приложение:

- Читает данные с COM-порта напрямую (через Python pyserial).

- Работает с локальными базами данных (SQLite) без ограничений.

- Управляет реле, моторами, источниками питания через интерфейс.

**Пример.** Приложение для калибровки датчиков давления. Оно должно опросить 10 датчиков по очереди, сравнить показания с эталоном и записать коэффициенты в память устройства. В браузере это невозможно. В приложении на Python Eel — легко.

### 1.3. Причина №2: Визуализация в реальном времени

Инженер мыслит графиками. Таблица из 1000 чисел ничего не скажет о тренде.

Современные веб-технологии (Chart.js, Plotly) позволяют строить красивые, интерактивные графики прямо внутри десктопного приложения.

Преимущества своего UI:

Мгновенная реакция: График обновляется по мере поступления данных.

Интерактивность: Можно выделить участок, приблизить, сохранить скриншот.

Аварийная сигнализация: Цвет меняется на красный, если значение превысило порог.

### 1.4. Причина №3: Автономная работа

Интернет — не всегда надёжен. Цех, подвал, полевые условия... Часто сеть отсутствует вообще.

Веб-сервис зависит от соединения. Десктопное приложение:

Работает полностью локально.

Хранит данные на устройстве.

Синхронизирует их с облаком только когда появляется сеть (если нужно).

**Пример.** Система мониторинга вибрации двигателя на удалённой насосной станции. Приложение пишет данные в локальный файл 24/7. Раз в сутки, когда появляется связь, оно отправляет отчёт на сервер.

### 1.5. Причина №4: Производительность и автоматизация

Рутина съедает время. Если вы каждый день запускаете тест по одному и тому же сценарию — автоматизируйте это.

Своё приложение позволяет:

Записать сценарий теста (нагреть до 50°C, ждать 5 мин, снять показания).

Запустить его одной кнопкой.

Сразу получать готовый отчёт (PDF/Excel).

### 1.6. Когда своё приложение — избыточно?

Честность важна. Писать ПО нужно не всегда.

Используйте готовые инструменты, если:

Нужно просто один раз снять показания (подойдёт мультиметр + блокнот).

Задача одноразовая (не стоит тратить 2 дня на код).

Есть готовое промышленное ПО (например, LabVIEW), которое уже куплено и настроено.

Но если ваш продукт — это инструмент для ежедневной работы, — своё приложение часто даёт лучший UX и экономию времени в долгосрочной перспективе.

### 1.7. Что мы узнали?

Готовые инструменты (Excel, Serial Monitor) часто ограничены для инженерных задач.

Ключевые преимущества своего ПО: доступ к железу, визуализация, автономность, автоматизация.

Современные фреймворки (Eel, Electron) позволяют использовать веб-технологии для создания удобного UI.

Выбор между «написать самому» и «взять готовое» — это вопрос частоты использования и сложности задачи.

**Чек-лист.** Практическое задание

Подумайте о трёх инструментах или процессах, которыми вы пользуетесь в работе (например: сбор данных с осциллографа, ведение журнала испытаний, управление нагревателем).

Для каждого ответьте:

2.a. Есть ли у него автоматизация?

2.b. Какие функции занимают больше всего времени вручную?

2.c. Какие из четырёх причин (доступ, визуализация, автономность, автоматизация) могли бы улучшить этот процесс?

Представьте, что вы создаёте приложение для одного из этих процессов.

Напишите 2–3 предложения: какую главную проблему решит ваше приложение?

**Пример.** «Я создаю панель для тестирования датчиков температуры. Сейчас я записываю показания вручную в журнал. Приложение позволит автоматически собирать данные с 4 датчиков одновременно, строить график и сохранять отчёт в CSV. Это сэкономит 30 минут на каждом тесте».

В следующей главе мы сравним традиционные подходы к созданию GUI (Tkinter, PyQt) с современными веб-обёртками, чтобы вы понимали все компромиссы и выбрали правильный инструмент.

## Обзор архитектур: традиционные GUI (Tkinter, PyQt) vs WEB-обёртки

О чём эта глава?

Вы уже поняли, зачем инженеру своё приложение. Теперь возникает главный вопрос: «На чём писать?»

Если вы знаете Python, логичным кажется использовать стандартные библиотеки для интерфейсов. Но есть нюанс: инженеру нужны не просто кнопки, а графики, логи и удобная вёрстка.

В этой главе мы сравним два подхода:

Традиционные GUI (Tkinter, PyQt) — классика Python-разработки.

Веб-обёртки (Eel, Electron) — современный гибрид Python + HTML/CSS.

Архитектурные компромиссы — что вы получаете и что теряете в каждом случае.

Почему для инженерных задач веб-технологии часто выигрывают.

Вы поймёте, почему в этой книге мы делаем ставку на связку Python + Web, и когда всё-таки стоит выбрать старый добрый Tkinter.

### 2.1. Классический путь: Tkinter и PyQt

Когда Python-разработчик слышит «нужен интерфейс», первая мысль — Tkinter. Он встроен в Python, не требует установки и работает «из коробки».

Пример на Tkinter:

```
python
1 import tkinter as tk
2
3 def on_click():
4     label.config(text="Данные получены!")
5
6 root = tk.Tk()
7 label = tk.Label(root, text="Ожидание...")
8 label.pack()
9 btn = tk.Button(root, text="Запросить", command=on_click)
10 btn.pack()
11 root.mainloop()
```

Рис. 2. Пример на Tkinter

Плюсы традиционных GUI:

Нативность: Приложение выглядит как часть ОС (особенно PyQt).

Лёгкость: Tkinter не тянет за собой браузер и весит килобайты.

Отсутствие сервера: Не нужно поднимать локальный веб-сервер, всё работает в одном процессе.

Минусы для инженера:

Визуализация: Построить красивый интерактивный график в Tkinter сложно (нужен matplotlib, который тяжёлый и не всегда интерактивный).

Вёрстка: Выравнивание элементов («сетка») в Tkinter часто превращается в боль. Сделать адаптивный интерфейс под разные экраны трудно.

Порог входа PyQt: PyQt мощнее, но у него сложный синтаксис (сигналы/слоты), и он требует изучения Qt-фреймворка.

**Инженерный вердикт.** Tkinter подойдёт для простой «кнопки вкл/выкл». Но если нужно отображать графики в реальном времени, логи и дашборды — вы потратите больше времени на борьбу с интерфейсом, чем на логику работы с железом.

## 2.2. Современный путь: Веб-обёртки (Hybrid Apps)

Идея проста: логика на Python, интерфейс в браузере.

Вы пишете backend на Python (работа с COM-портами, расчёты), а frontend — на HTML/CSS/JS (кнопки, графики Chart.js). Специальная оболочка (Eel, Electron) связывает их.

Пример архитектуры Eel:

```
python
1 # Python (backend)
2 import eel
3 eel.init('web')
4
5 @eel.expose
6 def get_sensor_data():
7     return 42.5 # Чтение из порта
8
9 eel.start('index.html')
```

Рис. 3. Пример архитектуры Eel

```
html
1 <!-- HTML/JS (frontend) -->
2 <div id="value">0</div>
3 <script>
4     async function update() {
5         let val = await eel.get_sensor_data();
6         document.getElementById('value').innerText = val;
7     }
8 </script>
```

Рис. 4. Пример архитектуры Eel (продолжение)

Плюсы веб-обёрток:

Визуализация: Библиотеки вроде Chart.js или Plotly позволяют строить потрясающие графики несколькими строками кода.

Вёрстка: CSS даёт полный контроль над дизайном. Легко сделать тёмную тему, адаптивность под планшеты.

Разделение ответственности: Инженер пишет Python, дизайнер (или готовый шаблон) отвечает за красоту.

Минусы:

Вес: Даже простое приложение тянет за собой движок браузера (хотя Eel использует системный браузер, что легче).

Сложность: Нужно знать основы HTML/JS (но для инженера это уровень «скопировать и поменять цифры»).

**Важно.** В этой книге мы используем Eel, потому что он максимально лёгкий для Python-разработчика. Electron мощнее, но требует знания Node.js, что избыточно для задач лаборатории.

## 2.3. Сравнительная таблица: Что выбрать инженеру?

Давайте посмотрим на цифры и факты. Представим, что мы создаём панель мониторинга температуры с графиком.

| Критерий           | Tkinter / PyQt            | Веб-обёртки (Eel)            | Комментарий                  |
|--------------------|---------------------------|------------------------------|------------------------------|
| Сложность старта   | ☆☆☆ (Средняя)             | ☆☆☆☆☆ (Очень низкая)         | Eel проще для новичка в GUI  |
| Красота интерфейса | ☆☆ (Устаревший вид)       | ☆☆☆☆☆ (Современный веб)      | Веб выглядит свежее          |
| Графики            | ☆☆ (Matplotlib)           | ☆☆☆☆☆ (Chart.js, Plotly)     | Веб-графики интерактивнее    |
| Вес приложения     | ☆☆☆☆☆ (Лёгкий)            | ☆☆☆ (Средний)                | Tkinter выигрывает в размере |
| Работа с COM       | ☆☆☆☆☆ (Прямая)            | ☆☆☆☆☆ (Прямая через Python)  | Здесь паритет                |
| Обучение           | Нужно учить GUI-фреймворк | Достаточно знать основы HTML | HTML знают многие студенты   |

Рис. 5.

Вывод: Если ваша цель — быстро сделать удобный инструмент для себя и коллег, веб-обёртки выигрывают за счёт готовых библиотек визуализации.

## 2.4. Почему мы выбираем Eel для этой книги?

В первой книге мы рассматривали Electron, NW.js и Eel на равных. Но для инженерных задач есть нюансы:

Python — язык инженера. Большинство скриптов для Arduino, обработки данных и расчётов уже написаны на Python. Eel позволяет использовать их без переписывания.

Electron избыточен. Он требует Node.js и собирает свой браузер. Для панели управления двигателем это как стрелять из пушки по воробьям.

Eel — это мост. Он не требует глубокого погружения в веб-разработку. Вы вызываете функции Python из JS так же легко, как из другого модуля Python.

**Наша философия.** Не становиться веб-разработчиком. Использовать веб-технологии как инструмент для отображения данных, оставляя Python для работы с железом.

## 2.5. Когда всё-таки стоит выбрать Tkinter?

Честность — важнее моды. Есть случаи, когда веб-обёртки не подходят:

Сверхлёгкие утилиты. Если нужно просто окно с одним полем ввода, которое должно весить 5 МБ.

Работа на старых ПК. В цехах иногда стоят компьютеры с 2 ГБ памяти. Запуск браузера может быть проблематичен.

Полная изоляция. Если политика безопасности запрещает запуск любых браузерных движков (редко, но бывает).

Но для 90% задач современной лаборатории — гибридный подход (Python + Web) является золотым стандартом.

Что мы узнали?

Традиционные GUI (Tkinter) легки, но сложны в вёрстке и визуализации.

Веб-обёртки дают современный интерфейс и мощные графики «из коробки».

Eel идеально подходит инженерам: логика на Python, интерфейс на HTML.

Выбор инструмента зависит от задачи: для дашбордов — веб, для микро-утилит — Tkinter.

В этой книге мы фокусируемся на Eel как на наиболее сбалансированном инструменте.

**Чек-лист.** Практическое задание

Найдите в интернете скриншоты интерфейсов на Tkinter и на веб-технологиях (например, дашборды Grafana или любые веб-приложения).

Сравните их визуально:

2.a. Где легче читать данные?

2.b. Где графики выглядят современнее?

Подумайте о вашем текущем проекте (из Главы 1):

→ Если бы вы делали его на Tkinter, сколько времени ушло бы на построение графика?

→ Если на Eel — сколько времени уйдёт на подключение Chart.js?

Напишите короткий вывод (2–3 предложения): почему для вашей задачи подходит гибридный подход?

**Пример.** «Для мониторинга вибрации мне нужно 4 графика в реальном времени. На Tkinter я бы потратил неделю на настройку matplotlib. На Eel я возьму готовый шаблон Chart.js за 1 час. Выбираю Eel».

В следующей главе мы познакомимся с инструментами поближе: Python Eel, Electron и Neutralino. Мы разберём их философию и поймём, почему Eel станет нашим основным рабочим инструментом.

## Знакомство с инструментами: Python Eel, Electron, Neutralino. Их философия и применение в инженерии

О чём эта глава?

Вы уже понимаете, зачем инженеру своё приложение и почему веб-технологии в оболочке — правильный выбор.

Теперь пришло время познакомиться с тремя главными инструментами этой книги:

Python Eel — минималистичный мост для Python-разработчиков.

Electron — промышленный стандарт для сложных продуктов.

Neutralino — лёгкая современная альтернатива без Node.js.

В этой главе вы узнаете:

**Заметка.** Какую проблему каждый из них решает в инженерном контексте.

**Заметка.** Какая философия лежит в основе каждого фреймворка.

**Заметка.** Кто их целевая аудитория — и почему вы среди них.

**Заметка.** Какие сильные и слабые стороны у каждого для задач лаборатории.

Эта глава поможет вам не просто «выбрать фреймворк», а понять, какой язык вам использовать.

### 3.1. Python Eel: «Мой скрипт заслуживает интерфейс»

История и философия

Eel родился в 2017 году из простой идеи:

*«Почему бы не использовать HTML/CSS/JS для интерфейса, а всю логику оставить на Python?»*

В отличие от Electron и Neutralino, Eel не встраивает браузер. Вместо этого он:

Запускает локальный веб-сервер на Python (через bottle или gevent-websocket).

Открывает окно системного браузера (Chrome, Edge, Safari).

Связывает Python и JavaScript через HTTP/WebSocket.

Философия Eel:

*Не становиться веб-разработчиком. Использовать веб-технологии как инструмент для отображения данных, оставляя Python для работы с железом.*

Целевая аудитория

| Кто вы                            | Почему Eel вам подходит                              |
|-----------------------------------|------------------------------------------------------|
| Инженер-экспериментатор           | Ваша логика уже на Python (pyserial, pandas, numpy)  |
| Исследователь/лаборант            | Нужен GUI за вечер, без изучения Node.js             |
| Студент технической специальности | Дипломный проект с реальным железом и современным UI |
| Data Scientist                    | Визуализация результатов модели в окне приложения    |

## Таблица 2. Целевая аудитория

**Заметка.** Сильные стороны для инженеров

Очень простой старт: 10 строк Python + 10 строк HTML = готовое приложение.

Полный доступ к экосистеме Python: pyserial, pandas, numpy, scikit-learn, opencv — всё работает «из коробки».

Минимальный порог входа: Не нужно знать JavaScript глубоко — достаточно базовых вызовов функций.

Идеален для скриптов с GUI: Панели управления, сбор данных, визуализация.

**Ошибка.** Слабые стороны

Нет встроенного доступа к нативным API ОС: Нет меню, троя, уведомлений «из коробки» (требуется доп. библиотеки вроде pystray).

Ограниченные возможности UI: Сложные интерфейсы (drag-and-drop, анимации) — сложнее, чем в Electron.

Производительность вызовов: Каждый вызов Python → JS проходит через HTTP/WebSocket (не критично для 95% задач).

Сборка в .exe: Требуется PyInstaller, что может быть громоздко с большими зависимостями.

**Пример из практики.** Панель мониторинга Arduino

Вот как выглядит типичное Eel-приложение для инженера (из ваших проектов):

```
python

1  # main.py
2  import eel
3  import serial
4  import threading
5
6  DEBUG_MODE = True  # Эмуляция без железа!
7
8  eel.init('www')
9
10 @eel.expose
11 def get_ports():
12     ports = serial.tools.list_ports.comports()
13     result = [port.device for port in ports]
14     if DEBUG_MODE:
15         result.append("EMULATOR")
16     return result
17
18 @eel.expose
19 def connect(port):
20     if DEBUG_MODE and port == "EMULATOR":
21         threading.Thread(target=emulate_data, daemon=True).start()
22         return {"status": "success"}
23     # Реальное подключение к COM-порту...
24
25 eel.start('index.html', size=(800, 600))
```

Рис. 6.

```
html
1 <!-- www/index.html -->
2 <div id="chart"></div>
3 <script src="chart.umd.min.js"></script>
4 <script>
5     async function update() {
6         let data = await eel.get_sensor_data();
7         // Обновление графика Chart.js
8     }
9 </script>
```

Рис. 7. Python Eel: «Мой скрипт заслуживает интерфейс» (продолжение)

Что это даёт инженеру:

Режим эмуляции — тестируйте код без Arduino

Графики в реальном времени — Chart.js + Python

Упаковка в .exe — PyInstaller, один файл для коллег

**Важно.** В этой книге мы делаем ставку на Eel как основной инструмент, потому что он максимально близок к мышлению инженера: Python для логики, HTML для красоты.

## 3.2. Electron: «Веб-приложение как десктопный продукт»

История и философия

Electron разработан в 2013 году в компании GitHub как внутренний инструмент для редактора Atom. Идея была простой:

*«Почему бы не использовать те же технологии, что и для веба, чтобы создавать десктопные приложения?»*

Философия Electron:

*Чёткое разделение ответственности: Main-процесс (Node.js) управляет ОС, Renderer-процесс (Chromium) отвечает за UI.*

Это делает архитектуру предсказуемой, масштабируемой и безопасной.

Целевая аудитория

| Кто вы                 | Почему Electron вам подходит                         |
|------------------------|------------------------------------------------------|
| Веб-разработчик        | Вы знаете JavaScript/TypeScript, React, Vue          |
| Команда продукта       | Строите сложный инструмент с долгосрочной поддержкой |
| Инженер с IoT-проектом | Нужен сложный UI + облачная синхронизация            |

Таблица 3. Целевая аудитория

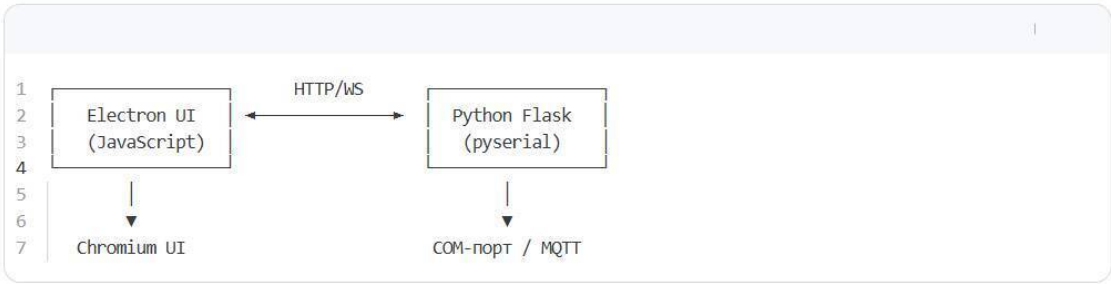
**Заметка.** Сильные стороны

Зрелая экосистема: electron-builder, DevTools, TypeScript-поддержка.

Чёткая архитектура с изоляцией процессов.

Огромное сообщество и документация.

- Полный доступ к API ОС: меню, трей, уведомления, диалоги — всё «из коробки».
- Поддержка всех современных веб-фреймворков: React, Vue, Svelte.
- Ошибка.** Слабые стороны для инженеров
- Высокое потребление памяти: Каждое окно — отдельный процесс Chromium (~150 МБ минимум).
- Большой размер дистрибутива: Минимум ~50–70 МБ даже для «Hello World».
- Сложность для новичков: Нужно понимать IPC, preload, sandbox.
- Требует знания Node.js: Если ваша логика на Python — придётся поднимать Flask-сервер отдельно.
- Пример из практики.** Electron + Python (Flask)
- Для инженеров Electron чаще всего используется в связке с Python-бэкендом:



- Рис. 8.
- Когда это оправдано:
  - Нужен очень сложный UI (редакторы, дашборды с десятками графиков).
  - Планируется масштабирование (команда разработчиков, версии, обновления).
  - Требуется нативная интеграция (трей, меню, автозапуск).
  - Важно.** Для большинства инженерных задач это избыточно. Electron — как стрелять из пушки по воробьям, если нужно просто читать данные с датчика.

### 3.3. Neutralino: «Лёгкий и современный»

- История и философия
- Neutralino — самый молодой из трёх (появился в 2020 году). Его миссия:  
«Создавать лёгкие кроссплатформенные приложения без зависимости от Node.js и тяжёлых браузерных движков.»
- Философия Neutralino:  
Использовать системный WebView (как Eel), но с нативным API и без Node.js.
- Целевая аудитория

| Кто вы                  | Почему Neutralino вам подходит          |
|-------------------------|-----------------------------------------|
| Инженер с ограничениями | Приложение должно весить < 20 МБ        |
| Разработчик утилит      | Нужен лёгкий инструмент для коллег      |
| IoT-энтузиаст           | Планируете запуск на слабых устройствах |

Таблица 4. Целевая аудитория

**Заметка.** Сильные стороны

Очень лёгкий: Дистрибутив ~2–5 МБ (в 10–20 раз меньше Electron!).

Не требует Node.js: Меньше зависимостей, проще сборка.

Системный WebView: Использует Edge WebView2 (Windows), WebKit (macOS), GTK (Linux).

Нативный API: Доступ к файловой системе, диалогам, трею — без дополнительных библиотек.

Безопасность: Песочница по умолчанию, нет Node.js в контексте.

**Ошибка.** Слабые стороны

Молодое сообщество: Меньше примеров, документации, готовых решений.

Ограниченный API: Не всё, что есть в Electron, доступно «из коробки».

Требуется знания JavaScript: Логика пишется на JS, Python подключается отдельно.

Меньше инструментов отладки: DevTools есть, но не такие мощные, как в Electron.

**Пример из практики.** Лёгкий монитор датчиков

```
javascript
1 // main.js (Neutralino)
2 Neutralino.filesystem.writeFile('data.csv', sensorData);
3 Neutralino.os.showMessageBox('Данные сохранены!', 'info');
```

Рис. 9. Лёгкий монитор датчиков

Когда это оправдано:

Нужно распространять среди коллег (маленький размер).

Запуск на слабых компьютерах (цех, лаборатория со старым парком).

Простой UI (кнопки, графики, логи) без сложных анимаций.

### 3.4. Сравнение в одном взгляде

| Критерий            | Python Eel                  | Electron                 | Neutralino        |
|---------------------|-----------------------------|--------------------------|-------------------|
| Основной язык       | Python                      | JavaScript/TypeScript    | JavaScript        |
| Архитектура         | Локальный сервер + браузер  | Chromium + Node.js       | Системный WebView |
| Размер дистрибутива | ~30–60 МБ*                  | ~50–150 МБ               | ~2–5 МБ           |
| Потребление RAM     | 80–150 МБ                   | 120–200 МБ               | 50–100 МБ         |
| Сложность старта    | ★★★★★ (Очень низкая)        | ★★★ (Средняя)            | ★★★★ (Низкая)     |
| Доступ к API ОС     | Ограниченный (через Python) | Полный                   | Полный (нативный) |
| Экосистема Python   | ✓ Полная                    | △ Через Flask/HTTP       | △ Через HTTP/CLI  |
| Идеальный сценарий  | Скрипт с GUI, Data Science  | Сложный продукт, команда | Лёгкая утилита    |

Таблица 5.

\* Размер Eel сильно зависит от Python-зависимостей: *pandas* + *numpy* → +50 МБ, *torch* → +500 МБ.

### 3.5. Когда что выбирать?

Инженерный чек-лист

**Выбирайте Python Eel, если.** Выбирайте Python Eel, если:

Ваша логика уже на Python (*pyserial*, *pandas*, *numpy*).

UI простой: кнопки, графики, таблицы, логи.

Пользователи — технические специалисты (коллеги, лаборанты).

Нужно быстро: прототип за вечер, диплом за неделю.

Нет требований к нативным меню/трею.

**Фокус.** Это выбор для 80% инженерных задач.

**Выбирайте Electron, если.** Выбирайте Electron, если:

Нужен очень сложный UI (редакторы, IDE, дашборды).

Планируется команда разработчиков и долгосрочная поддержка.

Нативная интеграция критична (трей, меню, уведомления, автозапуск).

Пользователи — массовый рынок (публичный продукт).

Вы знаете JavaScript лучше, чем Python.

**Выбирайте Neutralino, если.** Выбирайте Neutralino, если:

Размер дистрибутива критичен (< 20 МБ).

Запуск на слабых компьютерах (старые ПК в цеху, на производстве).

Простой UI, но нужна нативная интеграция (диалоги, файлы).

Хотите избежать зависимости от Node.js.

Готовы мириться с молодым сообществом.

## 3.6. Наша философия для этой книги

В первой книге мы рассматривали Electron, NW.js и Eel на равных. Но для инженерных задач есть нюансы:

| Аспект            | Первая книга                       | Эта книга                             |
|-------------------|------------------------------------|---------------------------------------|
| Основной фокус    | Electron (индустриальный стандарт) | Python Eel (язык инженера)            |
| Целевая аудитория | Веб-разработчики                   | Инженеры, исследователи, студенты     |
| Работа с железом  | Упоминалась                        | Центральная тема (COM, MQTT, Arduino) |
| NW.js             | Подробно рассмотрен                | Исключён (уступает Neutralino)        |
| IoT               | Не рассматривался                  | ESP32 + MQTT + Облако                 |

Рис. 10.

**Наша философия.** Не становиться веб-разработчиком. Использовать веб-технологии как инструмент для отображения данных, оставляя Python для работы с железом.

## 3.7. Реальный пример: От Arduino до облака

В этой книге мы пойдём по пути постепенного усложнения:

| Этап    | Проект               | Технологии                   | Для кого           |
|---------|----------------------|------------------------------|--------------------|
| Часть 2 | Калькулятор инженера | Eel + HTML/JS                | Новички            |
| Часть 3 | Мониторинг Arduino   | Eel + pyserial + Chart.js    | Инженеры с железом |
| Часть 3 | Эмулятор без Arduino | DEBUG_MODE + random          | Все (тесты/демо)   |
| Часть 4 | IoT-мониторинг ESP32 | MQTT + HiveMQ + GitHub Pages | Продвинутые        |
| Часть 5 | Electron + Flask     | Electron + Python backend    | Для сложных UI     |

Рис. 11. В этой книге мы пойдём по пути постепенного усложнения

Почему такой порядок?

Сначала Eel — потому что это самый быстрый способ для инженера получить результат. Потом COM-порты — потому что это основная задача лабораторных приложений. Затем IoT — потому что ESP32 открывает мир удалённого мониторинга.

В конце Electron — как альтернатива для тех, кому нужно больше возможностей.

Что мы узнали?

Eel — выбор для Python-разработчиков, которым нужен GUI без погружения в JavaScript.

Electron — выбор для продуктов, где важны масштабируемость, безопасность и команда.

Neutralino — выбор для лёгких утилит с минимальным размером дистрибутива.

Ни один из них не «лучше» — каждый решает свою задачу.

Для инженерных задач Eel часто оптимален — потому что Python уже является языком инженера.

**Чек-лист.** Практическое задание

Представьте три разных проекта:

А) Панель для сбора данных с 4 датчиков температуры (COM-порт, графики, сохранение в CSV).

В) Веб-дашборд для мониторинга оборудования в цеху (доступ из браузера, пользователи по всей стране).

С) Лёгкая утилита для калибровки прибора (вес < 20 МБ, запуск на старом ПК в лаборатории).

Для каждого проекта:

2.a. Выберите один из трёх фреймворков.

2.b. Обоснуйте выбор двумя причинами (например: «Eel, потому что основная логика на Python и UI простой»).

Подумайте: можно ли решить ту же задачу на другом фреймворке?

→ Какие сложности возникли бы?

**Пример ответа.** Пример ответа:

А) Eel — логика на Python (pyserial), UI простой (графики Chart.js), пользователи — лаборанты.

В) Electron + Flask — нужен доступ из браузера, сложная авторизация, массовые пользователи.

С) Neutralino — критичен размер, запуск на старом ПК, простой UI.

В следующей главе мы систематизируем всё, что узнали, и дадим вам чёткий алгоритм выбора: «Когда использовать Eel, Electron или Neutralino?» — с таблицей, примерами и контрольными вопросами.

## Критерии выбора: когда использовать какой фреймворк?

О чём эта глава?

Вы уже познакомились с тремя подходами:

Python Eel — для инженеров, живущих в экосистеме Python.

Electron — для сложных продуктов с мощным UI.

Neutralino — для лёгких утилит без лишних зависимостей.

Теперь настало время перейти от теории к решению. В этой главе вы узнаете:

**Заметка.** Как выбрать фреймворк на основе реальных требований проекта (железо, данные, пользователи).

**Заметка.** Какие пять ключевых вопросов задать себе перед стартом.

**Заметка.** Как сравнить фреймворки объективно — не по хайпу, а по цифрам и ограничениям (размер, СОМ-порты, сложность).

К концу главы у вас будет личная карта решений, которую вы сможете использовать в будущих проектах.

### 4.1. Пять вопросов перед выбором

Прежде чем писать код, ответьте на эти вопросы. Честные ответы сэкономят вам недели работы.

1. На каком языке написана основная логика?

Python? → Скорее всего, Eel.

Почему: pyserial, pandas, numpy уже у вас под рукой. Не нужно переписывать алгоритмы на JavaScript.

JavaScript/TypeScript? → Electron или Neutralino.

Почему: Если ваша логика уже на JS (например, веб-дашборд), нет смысла поднимать Python-сервер.

2. Насколько сложен пользовательский интерфейс?

Простые формы, кнопки, графики? → Eel подойдёт идеально.

Пример: Панель управления Arduino, мониторинг температуры.

Сложные анимации, drag-and-drop, редакторы кода? → Electron.

Пример: IDE, сложный конструктор схем.

Минимализм, стандартные элементы? → Neutralino.

3. Нужен ли доступ к нативным функциям ОС?

СОМ-порты, GPIO, файлы? → Eel (через Python) или Electron (через Node.js).

Нюанс: В Eel это одна строка `serial.Serial()`. В Electron — установка npm-пакетов и настройка preload.

Трей, системные уведомления, меню? → Electron (встроено).

Нюанс: В Eel потребуются доп. библиотеки (pystray), в Neutralino — нативный API.

4. Кто ваш пользователь?

Коллеги в офисе / Лаборанты? → Eel или Neutralino.

Почему: Можно собрать один .exe и забыть.

Миллионы пользователей (публичный продукт)? → Electron.

Почему: Нужны автообновления, кодовая подпись, защита от взлома, красивый установщик.

5. Каковы требования к размеру и производительности?

Установка должна быть < 20 МБ? → Neutralino.

Почему: Electron весит ~150 МБ даже для «Hello World». Eel зависит от библиотек (pandas + numpy = +100 МБ).

Память > 300 МБ — критично? → Избегайте Electron для простых задач.

Почему: Каждый процесс Electron — это отдельный Chromium.

## 4.2. Сравнительная таблица: объективные метрики

Давайте посмотрим на цифры. Представим, что мы создаём панель мониторинга датчиков с графиком в реальном времени.

| Критерий                 | Python Eel              | Electron                   | Neutralino                | Комментарий для инженера                         |
|--------------------------|-------------------------|----------------------------|---------------------------|--------------------------------------------------|
| Язык бэкенда             | Python 3.7+             | JavaScript/Node.js         | JavaScript                | Eel выигрывает, если логика на Python            |
| Доступ к COM-портам      | ✓ Прямой (pyserial)     | ⚠ Через serialport (npm)   | ⚠ Через плагин или бэкенд | Eel проще: import serial и всё работает          |
| Мин. размер дистрибутива | ~30–60 МБ*              | ~150 МБ                    | ~5–10 МБ                  | *Зависит от библиотек Python                     |
| Потребление RAM          | 80–150 МБ               | 150–250 МБ                 | 40–80 МБ                  | Neutralino самый лёгкий                          |
| Сложность старта         | ★★★★★ (Очень низкая)    | ★★★ (Средняя)              | ★★★★ (Низкая)             | Eel требует только pip install                   |
| Графики (Chart.js)       | ✓ Работает «из коробки» | ✓ Работает «из коробки»    | ✓ Работает «из коробки»   | Паритет: все используют веб-рендер               |
| Режим эмуляции           | ✓ Легко (Python-код)    | ⚠ Требуется моков в JS     | ⚠ Требуется моков в JS    | В Eel можно включить DEBUG_MODE одной переменной |
| IoT (MQTT)               | ✓ paho-mqtt (Python)    | ✓ mqtt.js (Node.js)        | ✓ mqtt.js (JS)            | Eel удобнее, если обработка данных на Python     |
| Упаковка (.exe)          | PyInstaller (просто)    | electron-builder (сложнее) | neu CLI (просто)          | Eel привычнее для Python-разработчика            |

Таблица 6.

**Важно.** Размер Eel сильно зависит от Python-зависимостей. Чистый скрипт — 30 МБ. С pandas + numpy — может вырасти до 150 МБ.

## 4.3. Алгоритм выбора: пошаговая инструкция

Текстовая версия алгоритма:

Шаг 1: Ваша логика на Python?

Да → Переходите к Шагу 2.

Нет → Смотрите в сторону Electron или Neutralino.

Шаг 2: Нужен ли сложный UI (анимации, редакторы)?

Нет (кнопки, графики, логи) → Python Eel.

Да → Electron (с Python-бэкендом через Flask/SocketIO).

Шаг 3: Критичен ли размер приложения (< 20 МБ)?

Да → Neutralino (но придётся писать логику на JS или поднимать Python-сервер отдельно).

Нет → Оставайтесь на Eel.

Шаг 4: Нужно ли работать с COM-портами?

Да → Eel даст самое простое решение (pyserial).

**Помните.** вы всегда можете комбинировать подходы. Например, тяжёлые вычисления — на Python (через subprocess из Electron), UI — на React. Но для инженерных задач это часто избыточно.

## 4.4. Когда НИ ОДИН из трёх не подходит?

Иногда правильный ответ — не использовать ни один из них. Это нормально!

Рассмотрите альтернативы, если:

Приложение должно весить менее 5 МБ → Смотрите в сторону Tauri (Rust + WebView).

Плюс: Очень лёгкий, безопасный.

Минус: Требуется знания Rust для бэкенда.

Требуется максимальная производительность (видео, 3D) → Нативная разработка (C++, C#, Qt).

Плюс: Полный контроль над железом.

Минус: Высокий порог входа, долго писать.

Вы делаете мобильное приложение → Electron/Eel/Neutralino не работают на iOS/Android.

Решение: Flutter, React Native или Kivy (Python).

**Примечание.** Tauri — достойный конкурент Electron с размером дистрибутива ~3–5 МБ, но он требует знания Rust. Мы не включаем его в основную триаду, так как наша цель — Python и веб-технологии для инженеров.

## 4.5. Частые ловушки новичков

| ОШИБКА                                                      | ПОСЛЕДСТВИЕ                                     | КАК ИЗБЕЖАТЬ                                                        |
|-------------------------------------------------------------|-------------------------------------------------|---------------------------------------------------------------------|
| Выбрать Electron для простого «считывателя СОМ-порта»       | Пользователь скачивает 150 МБ ради одной кнопки | Спросите: «Действительно ли нужен Electron?»                        |
| Ожидать от Eel нативного меню «из коробки»                  | Разочарование, поиск несуществующих функций     | Знайте ограничения: Eel — не замена Electron                        |
| Использовать <code>time.sleep()</code> в главном потоке Eel | Интерфейс зависает при чтении данных            | Используйте <code>threading</code> или <code>asyncio</code>         |
| Собирать Eel-приложение без виртуального окружения          | Конфликты библиотек, огромные размеры           | Всегда используйте <code>venv</code> перед <code>PyInstaller</code> |
| Забывать про режим эмуляции ( <code>DEBUG_MODE</code> )     | Невозможность тестов без железа                 | Внедряйте эмуляцию с первого дня разработки                         |

Таблица 7.

## 4.6. Наш выбор для этой книги

В первой книге мы рассматривали Electron, NW.js и Eel на равных. Но для инженерных задач есть нюансы:

| Аспект            | Первая книга                       | Эта книга                             |
|-------------------|------------------------------------|---------------------------------------|
| Основной фокус    | Electron (индустриальный стандарт) | Python Eel (язык инженера)            |
| Целевая аудитория | Веб-разработчики                   | Инженеры, исследователи, студенты     |
| Работа с железом  | Упоминалась                        | Центральная тема (СОМ, MQTT, Arduino) |
| NW.js             | Подробно рассмотрен                | Исключён (уступает Neutralino/Eel)    |
| IoT               | Не рассматривался                  | ESP32 + MQTT + Облако                 |

Таблица 8.

**Наша философия.** Не становиться веб-разработчиком. Использовать веб-технологии как инструмент для отображения данных, оставляя Python для работы с железом.

Что мы узнали?

Выбор фреймворка — это анализ требований, а не следование тренду.

Язык бэкенда — главный фильтр: Python → Eel, JS → Electron/Neutralino.

Eel оптимален для инженерных задач (СОМ-порты, данные, Python-библиотеки).

Electron нужен для сложных продуктов с массовым пользователем.

Neutralino — выбор для лёгких утилит с ограничением по размеру.

Всегда задавайте себе пять ключевых вопросов перед стартом.

**Чек-лист.** Практическое задание

Опишите свой реальный или гипотетический проект (1–2 предложения).

Пример: «Инструмент для анализа логов сервера с графиками и фильтрацией».

Пройдите по алгоритму выбора из раздела 4.3.

Заполните таблицу:

| КРИТЕРИЙ                               | ВАШ ОТВЕТ | ПОДХОДЯЩИЙ ФРЕЙМВОРК |
|----------------------------------------|-----------|----------------------|
| Язык бэкенда                           | ?         | ?                    |
| Сложность UI                           | ?         | ?                    |
| Нужны ли нативные API?<br>(COM, файлы) | ?         | ?                    |
| Целевая аудитория                      | ?         | ?                    |
| Ограничения по размеру?                | ?         | ?                    |

Таблица 9.

Сделайте вывод: почему именно этот фреймворк — лучший выбор?

**Пример заполнения.** Пример заполнения:

Проект: «Панель управления нагревателем через Arduino».

Бэкенд: Python (pyserial, threading).

UI: Графики температуры, кнопки вкл/выкл.

API: COM-порт обязательно.

Вывод: Eel — идеален, так как позволяет использовать всю мощь Python для работы с портом и быстро сделать GUI на HTML/Chart.js. Electron избыточен, Neutralino потребует лишней настройки связи с Python.

Теперь вы готовы к практической части книги.

В Части 2 мы углубимся в Python Eel: от установки до создания первого приложения с графиками.

В Части 3 подключим Arduino и научимся работать с COM-портами.

В Части 4 выйдем в IoT с ESP32 и MQTT.

Выбор сделан — пора писать код!

## Часть 2. Python Eel — основной инструмент инженера

### Быстрый старт с Eel. Установка, структура проекта, первый запуск

О чём эта глава?

Вы выбрали Python Eel — отличное решение для инженера, который хочет быстро создать интерфейс для своего скрипта.

Теперь пришло время перейти от теории к практике. В этой главе вы:

Установите Eel и проверите работоспособность.

Создадите структуру проекта «правильно» с первого дня.

Напишете первое приложение за 10 минут.

Запустите его и увидите результат в окне браузера.

Поймёте, как Python и JavaScript «общаются» между собой.

К концу главы у вас будет рабочее приложение, которое можно развивать дальше — добавлять графики, кнопки, работу с COM-портами.

#### 5.1. Почему Eel? Философия быстрого старта

Прежде чем писать код, давайте ответим на вопрос: почему именно Eel?

| Фреймворк  | Время до первого окна | Порог входа             | Идеально для                          |
|------------|-----------------------|-------------------------|---------------------------------------|
| Tkinter    | 15 минут              | Средний                 | Простые утилиты                       |
| PyQt       | 1–2 часа              | Высокий                 | Сложные коммерческие продукты         |
| Electron   | 2–3 часа              | Высокий (нужен Node.js) | Веб-разработчики                      |
| Python Eel | 10 минут              | Очень низкий            | Инженеры, Data Science, автоматизация |

*Таблица 10.*

**Наша философия.** Инженеру не нужно становиться веб-разработчиком. Ему нужно быстро обернуть свой Python-скрипт в удобный интерфейс. Eel даёт именно это.

Eel — это мост, а не замена.

Python остаётся для логики: расчёты, COM-порты, файлы, данные.

HTML/CSS/JS остаются для красоты: кнопки, графики, вёрстка.

Eel связывает их двумя простыми вызовами функций.

#### 5.2. Установка Eel: Пошагово

Eel устанавливается через стандартный менеджер пакетов Python — pip.

**Шаг 1. Проверка Python**

Убедитесь, что у вас установлен Python 3.7 или выше:

```
bash
1 python --version
2 # или
3 python3 --version
```

Рис. 12. Убедитесь, что у вас установлен Python 3.7 или выше

**Важно.** Если у вас несколько версий Python, используйте `py -3 --version` (Windows) или `python3 --version` (macOS/Linux).

### Шаг 2. Создание виртуального окружения (рекомендуется)

Это хорошая практика — изолировать зависимости проекта:

```
bash
1 # Создание виртуального окружения
2 python -m venv venv
3
4 # Активация (Windows)
5 venv\Scripts\activate
6
7 # Активация (macOS/Linux)
8 source venv/bin/activate
```

Рис. 13. Это хорошая практика — изолировать зависимости проекта

**Совет.** Зачем это нужно? Когда вы будете упаковывать приложение в .exe (Глава 8), PyInstaller соберёт только те библиотеки, которые установлены в виртуальном окружении. Это уменьшит размер дистрибутива и избежит конфликтов.

### Шаг 3. Установка Eel

```
bash
1 pip install eel
```

Рис. 14.

### Шаг 4. Проверка установки

Создайте файл `test.py` и выполните:

```
python
1 import eel
2 print(f"Eel версия: {eel.__version__}")
```

Рис. 15. Создайте файл `test.py` и выполните

Если нет ошибок — установка успешна!

### 5.3. Структура проекта: Правильно с первого дня

Одна из главных ошибок новичков — сваливать все файлы в одну кучу.

Правильная структура для Eel-приложения:

```
1 my_engineering_app/
2   |— main.py           # Точка входа (Python-бэкенд)
3   |— requirements.txt  # Список зависимостей
4   |— www/              # Папка с фронтендом (HTML/CSS/JS)
5       |— index.html    # Главный экран
6       |— style.css     # Стили (внешний вид)
7       |— app.js        # Логика интерфейса
8       |— chart.umd.min.js # Библиотеки (графики и т.д.)
9   |— assets/           # Иконки, изображения (опционально)
10  |— data/              # Логи, конфиги, отчёты (опционально)
```

Рис. 16. Правильная структура для Eel-приложения

Ключевое правило: Все HTML/CSS/JS-файлы должны лежать в папке, которую вы укажете в `eel.init()`. По умолчанию это папка `www`.

Почему такая структура?

| Папка   | Зачем нужна                                          |
|---------|------------------------------------------------------|
| www/    | Eel ищет здесь интерфейс. Отдельная папка = порядок. |
| main.py | Логика приложения. Не смешивайте с фронтендом.       |
| assets/ | Иконки, логотипы. Легко найти при сборке.            |
| data/   | Данные, которые создаёт приложение (логи, CSV).      |

Таблица 11. Почему такая структура?

### 5.4. Первое приложение: «Калькулятор инженера»

Создадим простое приложение, которое показывает мощь связи между Python и JavaScript.

Файл 1: `main.py` (Бэкенд)

```
python

1 import eel
2 import os
3
4 # === ИНИЦИАЛИЗАЦИЯ EEL ===
5 # Указываем папку с фронтендом
6 eel.init(os.path.join(os.path.dirname(__file__), 'www'))
7
8 # === ФУНКЦИИ PYTHON, ДОСТУПНЫЕ ИЗ JS ===
9 @eel.expose
10 def calculate_power(voltage, current):
11     """Расчёт мощности:  $P = U \cdot I$ """
12     try:
13         voltage = float(voltage)
14         current = float(current)
15         power = voltage * current
16         return {
17             "status": "success",
18             "power": round(power, 2),
19             "message": f"Мощность: {power:.2f} Вт"
20         }
21     except Exception as e:
22         return {
23             "status": "error",
24             "message": f"Ошибка: {str(e)}"
25         }
```

Рис. 17. Файл 1: main.py (Бэкенд)

```
python

1 import eel
2 import os
3
4 # === ИНИЦИАЛИЗАЦИЯ EEL ===
5 # Указываем папку с фронтендом
6 eel.init(os.path.join(os.path.dirname(__file__), 'www'))
7
8 # === ФУНКЦИИ PYTHON, ДОСТУПНЫЕ ИЗ JS ===
9 @eel.expose
10 def calculate_power(voltage, current):
11     """Расчёт мощности:  $P = U \cdot I$ """
12     try:
13         voltage = float(voltage)
14         current = float(current)
15         power = voltage * current
16         return {
17             "status": "success",
18             "power": round(power, 2),
19             "message": f"Мощность: {power:.2f} Вт"
20         }
21     except Exception as e:
22         return {
23             "status": "error",
24             "message": f"Ошибка: {str(e)}"
25         }
```

Рис. 18. Файл 1: main.py (Бэкенд) (продолжение)

```
26
27 @eel.expose
28 def get_greeting(name):
29     """Приветствие пользователя"""
30     return f"Привет, {name}! Добро пожаловать в инженерное приложение."
31
32 # === ЗАПУСК ПРИЛОЖЕНИЯ ===
33 if __name__ == '__main__':
34     # mode='chrome' — открыть в браузере Chrome
35     # mode='edge' — открыть в Edge (Windows)
36     # size=(width, height) — размер окна
37     eel.start('index.html', size=(600, 500), position=(100, 100))
```

Рис. 19. Файл 1: main.py (Бэкенд) (продолжение)

**Обратите внимание.** Декоратор @eel.expose делает функцию доступной из JavaScript. Без него функция останется «внутри» Python.

Файл 2: www/index.html (Фронтенд)

```
html
1 <!DOCTYPE html>
2 <html lang="ru">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Калькулятор инженера</title>
7
8     <!-- Подключаем стили -->
9     <link rel="stylesheet" href="style.css">
10
11     <!-- Eel автоматически генерирует /eel.js -->
12     <script type="text/javascript" src="/eel.js"></script>
13 </head>
14 <body>
15     <div class="container">
16         <h1>🔧 Калькулятор инженера</h1>
17
18         <div class="form-group">
19             <label for="voltage">Напряжение (В):</label>
20             <input type="number" id="voltage" placeholder="220" value="220">
21         </div>
22     </div>
```

Рис. 20. Файл 2: www/index.html (Фронтенд)

```
23     <div class="form-group">
24         <label for="current">Ток (А):</label>
25         <input type="number" id="current" placeholder="5" value="5">
26     </div>
27
28     <button onclick="calculate()">Рассчитать мощность</button>
29
30     <div id="result" class="result">
31         Результат появится здесь
32     </div>
33
34     <div id="greeting" class="greeting"></div>
35 </div>
36
37 <!-- Подключаем логику интерфейса -->
38 <script src="app.js"></script>
39 </body>
40 </html>
```

Рис. 21. Файл 2: `www/index.html` (Фронтенд) (продолжение)

```
html
1 <!DOCTYPE html>
2 <html lang="ru">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Калькулятор инженера</title>
7
8     <!-- Подключаем стили -->
9     <link rel="stylesheet" href="style.css">
10
11     <!-- Eel автоматически генерирует /eel.js -->
12     <script type="text/javascript" src="/eel.js"></script>
13 </head>
14 <body>
15     <div class="container">
16         <h1>🔧 Калькулятор инженера</h1>
17
18         <div class="form-group">
19             <label for="voltage">Напряжение (В):</label>
20             <input type="number" id="voltage" placeholder="220" value="220">
21         </div>
22     </div>
```

Рис. 22. Файл 2: `www/index.html` (Фронтенд) (продолжение)

Файл 3: `www/style.css` (Стили)

```
css

1 body {
2   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
3   background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
4   display: flex;
5   justify-content: center;
6   align-items: center;
7   min-height: 100vh;
8   margin: 0;
9 }
10
11 .container {
12   background: white;
13   padding: 30px;
14   border-radius: 15px;
15   box-shadow: 0 10px 30px rgba(0,0,0,0.3);
16   width: 400px;
17   text-align: center;
18 }
19
20 h1 {
21   color: #333;
22   margin-bottom: 25px;
23 }
24
```

Рис. 23. Файл 3: `www/style.css` (Стили)

```

25 .form-group {
26   margin-bottom: 15px;
27   text-align: left;
28 }
29
30 label {
31   display: block;
32   margin-bottom: 5px;
33   color: #555;
34   font-weight: bold;
35 }
36
37 input {
38   width: 100%;
39   padding: 10px;
40   border: 2px solid #ddd;
41   border-radius: 8px;
42   font-size: 16px;
43   box-sizing: border-box;
44 }
45
46 input:focus {
47   border-color: #667eea;
48   outline: none;
49 }
50
```

Рис. 24. Файл 3: `www/style.css` (Стили) (продолжение)

```
51 button {
52     background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
53     color: white;
54     border: none;
55     padding: 12px 30px;
56     border-radius: 8px;
57     font-size: 16px;
58     cursor: pointer;
59     width: 100%;
60     margin-top: 10px;
61     transition: transform 0.2s;
62 }
63
64 button:hover {
65     transform: scale(1.02);
66 }
67
68 .result {
69     margin-top: 20px;
70     padding: 15px;
71     background: #f8f9fa;
72     border-radius: 8px;
73     font-size: 18px;
74     font-weight: bold;
75     color: #333;
76 }
77
```

Рис. 25. Файл 3: `www/style.css` (Стили) (продолжение)

```
78 .greeting {
79     margin-top: 15px;
80     color: #666;
81     font-style: italic;
82 }
```

Рис. 26. Файл 3: `www/style.css` (Стили) (продолжение)

```
javascript

1 // === ВЫЗОВ PYTHON ИЗ JAVASCRIPT ===
2 async function calculate() {
3     // Получаем значения из полей ввода
4     const voltage = document.getElementById('voltage').value;
5     const current = document.getElementById('current').value;
6
7     // Вызываем Python-функцию через Eel
8     // Обратите внимание на ДВЕ пары скобок: eel.func()()
9     const result = await eel.calculate_power(voltage, current)();
10
11     // Отображаем результат
12     const resultDiv = document.getElementById('result');
13     if (result.status === 'success') {
14         resultDiv.innerHTML = `⚡ ${result.message}`;
15         resultDiv.style.color = '#28a745';
16     } else {
17         resultDiv.innerHTML = `❌ ${result.message}`;
18         resultDiv.style.color = '#dc3545';
19     }
20 }
21
22 // === ВЫЗОВ ИЗ PYTHON ПРИ ЗАПУСКЕ ===
23 // Эту функцию вызовет Python при старте
24 eel.expose(function show_greeting(name) {
25     document.getElementById('greeting').innerText = name;
26 });
27
```

Рис. 27. Файл 3: `www/style.css` (Стили) (продолжение)

Файл 4: `www/app.js` (Логика интерфейса)

```
28 // Приветствие при загрузке страницы
29 window.onload = async function() {
30     const greeting = await eel.get_greeting("Инженер")();
31     document.getElementById('greeting').innerText = greeting;
32 };
```

Рис. 28. Файл 4: `www/app.js` (Логика интерфейса)

## 5.5. Запуск приложения

### Шаг 1. Запуск через терминал

```
bash

1 python main.py
```

Рис. 29.

### Шаг 2. Что произойдёт?

Python запустит локальный веб-сервер (обычно на `http://localhost:8000`).

Откроется окно браузера с вашим интерфейсом.

Вы увидите форму с полями «Напряжение» и «Ток».



| Режим          | Описание                             | Когда использовать                        |
|----------------|--------------------------------------|-------------------------------------------|
| mode='chrome'  | Открывает в Google Chrome            | Кроссплатформенно, если Chrome установлен |
| mode='edge'    | Открывает в Microsoft Edge           | Windows 10/11 (рекомендуется)             |
| mode='firefox' | Открывает в Firefox                  | Если пользователь предпочитает Firefox    |
| mode='default' | Системный браузер по умолчанию       | Максимальная совместимость                |
| mode=False     | Не открывать браузер (только сервер) | Для отладки или веб-доступа               |

Таблица 12. Eel поддерживает несколько режимов открытия окна

Пример с параметрами:

```
python
1 eel.start(
2     'index.html',
3     size=(800, 600),           # Размер окна (ширина, высота)
4     position=(100, 100),      # Позиция окна (x, y)
5     fullscreen=False,         # Полный экран
6     close_callback=on_close   # Функция при закрытии окна
7 )
8
9 def on_close(page, sockets):
10     print("Приложение закрыто!")
11     # Здесь можно сохранить данные, закрыть сокет-порт и т.д.
```

Рис. 31. Пример с параметрами

## 5.8. Частые ошибки и решения

| ПРОБЛЕМА                        | ПРИЧИНА                               | РЕШЕНИЕ                                                                  |
|---------------------------------|---------------------------------------|--------------------------------------------------------------------------|
| Белый экран                     | Неправильный путь к <code>www/</code> | Проверьте <code>eel.init('www')</code>                                   |
| <code>eel</code> is not defined | Не подключён <code>/eel.js</code>     | Добавьте <code>&lt;script src="/eel.js"&gt;&lt;/script&gt;</code> в HTML |
| Функция не вызывается           | Забыли <code>@eel.expose</code>       | Добавьте декоратор в Python                                              |
| Две пары скобок?                | Синтаксис Eel                         | <code>eel.func()()</code> — это нормально!                               |
| Порт занят                      | Другой Eel запущен                    | Закройте предыдущий процесс или укажите <code>port=8081</code>           |

Таблица 13.

## 5.9. Подготовка к следующему шагу: Режим эмуляции

В следующих главах мы будем работать с COM-портами и Arduino. Но что если у вас нет устройства под рукой?

Мы внедрим режим эмуляции (DEBUG\_MODE) с самого начала.

Вот как это будет выглядеть в `main.py`:

```
python
1 # === НАСТРОЙКИ ===
2 DEBUG_MODE = True # ← Переключите на False, когда подключите Arduino
3
4 @eel.expose
5 def get_ports():
6     ports = serial.tools.list_ports.comports()
7     result = [port.device for port in ports]
8     if DEBUG_MODE:
9         result.append("EMULATOR") # Добавляем опцию эмулятора
10    return result
```

Рис. 32. Вот как это будет выглядеть в `main.py`

**Важно.** Мы вернёмся к этому в Главе 10, но уже сейчас вы понимаете философию: тестируйте без железа, развёртывайте с железом.

Что мы узнали?

Eel устанавливается за 1 минуту через `pip install eel`.

Структура проекта: `main.py` + папка `www/` с HTML/CSS/JS.

Связка Python ↔ JS: декоратор `@eel.expose` + вызов `eel.func()()`.

Первое приложение работает за 10 минут без глубоких знаний веба.

Режимы запуска позволяют выбрать браузер и размер окна.

Виртуальное окружение — хорошая практика для будущей упаковки в `.exe`.

**Чек-лист.** Практическое задание

Создайте проект по структуре из раздела 5.3.

Запустите калькулятор из этой главы.

Модифицируйте приложение:

3.a. Добавьте третье поле: «Сопротивление (Ом)».

3.b. Создайте Python-функцию `calculate_resistance(voltage, current)`, которая возвращает сопротивление по закону Ома:  $R = U / I$ .

3.c. Добавьте кнопку «Рассчитать сопротивление» и отобразите результат.

Поэкспериментируйте с режимом запуска:

4.a. Измените размер окна на (800, 600).

4.b. Попробуйте `mode='chrome'` и `mode='edge'` — есть ли разница?

Добавьте обработку ошибок:

5.a. Что будет, если ввести 0 в поле тока при расчёте сопротивления?

5.b. Реализуйте проверку в Python и верните понятное сообщение об ошибке.

**Совет.** Пример решения для 3.b:

```
python
1 @eel.expose
2 def calculate_resistance(voltage, current):
3     try:
4         voltage = float(voltage)
5         current = float(current)
6         if current == 0:
7             return {"status": "error", "message": "Ток не может быть нулём!"}
8         resistance = voltage / current
9         return {"status": "success", "resistance": round(resistance, 2)}
10    except Exception as e:
11        return {"status": "error", "message": str(e)}
```

Рис. 33. Пример решения для 3.b

В следующей главе мы углубимся в механизм вызовов: как передавать сложные данные (списки, словари), как вызывать JavaScript из Python и как обрабатывать ошибки. Вы научитесь строить двусторонний мост между бэкендом и фронтендом.

## Связка Python и JS. Вызов функций, передача данных, асинхронность без блокировки UI

О чём эта глава?

Вы уже создали первое приложение на Eel и увидели, как Python и JavaScript могут работать вместе. Но настоящее мастерство — в понимании того, как они общаются.

В этой главе вы узнаете:

Как вызывать Python-функции из JavaScript (и наоборот) с правильной синхронизацией.

Как передавать сложные данные — списки, словари, JSON-объекты.

Почему UI зависает при чтении COM-порта и как этого избежать.

Как использовать `threading` (многопоточность) для фоновых\* задач без блокировки интерфейса.

Как обрабатывать ошибки на обеих сторонах моста.

**П р и м е ч а н и е\*** - фоновые задачи – это операции, которые не должны блокировать основной поток (особенно графический интерфейс). Например: загрузка файлов из сети, обработка больших данных, отправка запросов к API, логирование (процесс регистрации данных), мониторинг состояния системы.

К концу главы вы сможете создать приложение, которое читает данные с датчиков в реальном времени, не замораживая интерфейс.

### 6.1. Двусторонний мост: как работает связь

Eel создаёт мост между Python и JavaScript через локальный веб-сервер. Когда вы вызываете функцию из JS в Python (или наоборот), запрос проходит через этот сервер.

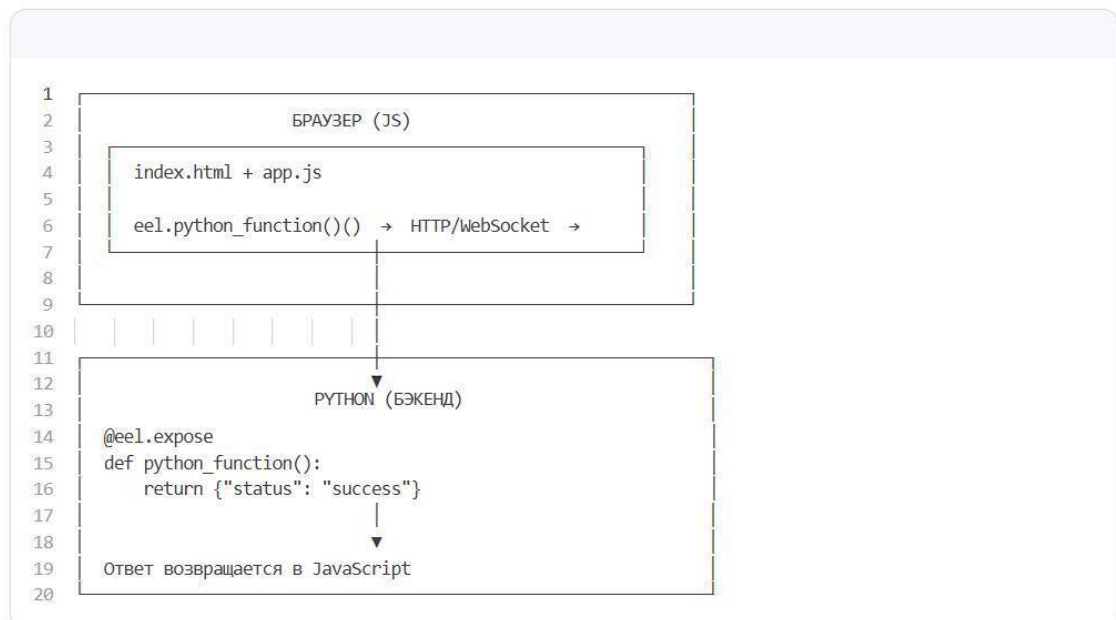


Рис. 34.

Ключевой момент: Eel использует асинхронные вызовы. Это значит, что JavaScript не ждёт ответа Python — он продолжает работать. Но для получения результата нужно использовать `await`.

## 6.2. Вызов Python из JavaScript: правильный синтаксис

Базовый пример

Вот как выглядит типичный вызов Python-функции из JavaScript:

```
python
1 # main.py
2 import eel
3
4 eel.init('www')
5
6 @eel.expose
7 def calculate_power(voltage, current):
8     """Расчёт мощности:  $P = U \cdot I$ """
9     return voltage * current
10
11 eel.start('index.html')
```

```
javascript
1 // www/app.js
2 async function calculate() {
3     // Обратите внимание на ДВЕ пары скобок!
4     const result = await eel.calculate_power(220, 5)();
5     console.log("Мощность:", result);
6 }
```

Рис. 35. Вот как выглядит типичный вызов Python-функции из JavaScript

Почему две пары скобок?

Это частый вопрос новичков. Давайте разберёмся:

| Синтаксис                 | Что делает                                                      |
|---------------------------|-----------------------------------------------------------------|
| <code>eel.func</code>     | Ссылка на функцию (не вызывает)                                 |
| <code>eel.func()</code>   | Первая пара — передаёт аргументы в Python                       |
| <code>eel.func()()</code> | Вторая пара — выполняет вызов и возвращает <code>Promise</code> |

Рис. 36. Это частый вопрос новичков. Давайте разберёмся

**Запомните.** `await eel.function_name(args)()` — это стандартный паттерн Eel.

Обработка ошибок

Python-исключения передаются в JavaScript как ошибки `Promise`:

```
python
1 # main.py
2 @eel.expose
3 def divide(a, b):
4     if b == 0:
5         raise ValueError("Деление на ноль!")
6     return a / b
```

Рис. 37. Python-исключения передаются в JavaScript как ошибки Promise

```
javascript
1 // www/app.js
2 async function safeDivide() {
3     try {
4         const result = await eel.divide(10, 0)();
5         console.log("Результат:", result);
6     } catch (error) {
7         console.error("Ошибка:", error);
8         // Показываем пользователю понятное сообщение
9         document.getElementById('error').innerText = "Деление на ноль запрещено!";
10    }
11 }
```

Рис. 38. Python-исключения передаются в JavaScript как ошибки Promise (продолжение)

### 6.3. Передача сложных данных

Eel поддерживает все типы, которые можно сериализовать в JSON:

| Python                | JavaScript              |
|-----------------------|-------------------------|
| str, int, float, bool | string, number, boolean |
| list                  | Array                   |
| dict                  | Object                  |
| None                  | null                    |

Рис. 39. Eel поддерживает все типы, которые можно сериализовать в...

Пример: передача словаря

python

```
1 # main.py
2 @eel.expose
3 def get_sensor_config():
4     """Возвращает конфигурацию датчика"""
5     return {
6         "name": "Термопара К-типа",
7         "min_temp": -200,
8         "max_temp": 1350,
9         "precision": 0.1,
10        "calibrated": True,
11        "ports": ["COM1", "COM3", "COM5"]
12    }
```

Рис. 40. Пример: передача словаря

javascript

```
1 // www/app.js
2 async function loadConfig() {
3     const config = await eel.get_sensor_config();
4
5     document.getElementById('sensor-name').innerText = config.name;
6     document.getElementById('temp-range').innerText =
7         `${config.min_temp}°C ... ${config.max_temp}°C`;
8
9     // Работаем с массивом портов
10    config.ports.forEach(port => {
11        const option = document.createElement('option');
12        option.value = port;
13        option.innerText = port;
14        document.getElementById('port-select').appendChild(option);
15    });
16 }
```

Рис. 41. Пример: передача словаря (продолжение)

Пример: передача списка данных для графика

python

```
1 # main.py
2 @eel.expose
3 def get_temperature_history():
4     """Возвращает историю температур за последний час"""
5     import random
6     from datetime import datetime, timedelta
7
8     now = datetime.now()
9     data = []
10
11     for i in range(60):
12         timestamp = (now - timedelta(minutes=i)).strftime("%H:%M")
13         temp = round(25 + random.random() * 10, 1)
14         data.append({"time": timestamp, "temp": temp})
15
16     return data # Список словарей
```

Рис. 42. Пример: передача списка данных для графика

javascript

```
1 // www/app.js
2 async function drawChart() {
3     const history = await eel.get_temperature_history();
4
5     // Подготовка данных для Chart.js
6     const labels = history.map(item => item.time);
7     const values = history.map(item => item.temp);
8
9     // Отрисовка графика (подробнее в Главе 7)
10    const ctx = document.getElementById('tempChart').getContext('2d');
11    new Chart(ctx, {
12        type: 'line',
13        data: {
14            labels: labels.reverse(),
15            datasets: [{
16                label: 'Температура (°C)',
17                data: values.reverse(),
18                borderColor: 'rgb(75, 192, 192)',
19                tension: 0.1
20            }]
21        }
22    });
23 }
```

Рис. 43. Пример: передача списка данных для графика (продолжение)

## 6.4. Вызов JavaScript из Python

Иногда нужно обновить интерфейс из Python — например, при поступлении новых данных с датчика.

Базовый пример

python

```
1 # main.py
2 import eel
3
4 eel.init('www')
5
6 @eel.expose
7 def start_measurement():
8     """Запускает измерение и обновляет UI"""
9     # Вызываем JS-функцию для обновления прогресса
10    eel.update_status("Измерение начато")()
11
12    # Имитация работы
13    import time
14    time.sleep(2)
15
16    # Обновляем результат
17    eel.update_result(42.5)()
18
19 eel.start('index.html')
```

Рис. 44. Базовый пример

javascript

```
1 // www/app.js
2 // Эти функции должны быть глобальными!
3 eel.expose(function update_status(message) {
4     document.getElementById('status').innerText = message;
5 });
6
7 eel.expose(function update_result(value) {
8     document.getElementById('result').innerText = value + "°C";
9 });
```

Рис. 45. Базовый пример (продолжение)

**Важно.** JS-функции, вызываемые из Python, должны быть глобальными (в window). Не используйте const или let — только function или window.functionName.

## 6.5. Асинхронность: как не заблокировать UI

Это самая важная тема для инженерных приложений.

Проблема: блокировка при чтении COM-порта

Представьте, что вы читаете данные с Arduino:

```
python
1 # ❌ ПЛОХО: Блокирует UI
2 @eel.expose
3 def read_sensor():
4     import serial
5     ser = serial.Serial('COM3', 9600)
6     data = ser.readline() # Блокирует поток!
7     ser.close()
8     eel.update_data(data)() # UI обновится только после чтения
```

Рис. 46. Представьте, что вы читаете данные с Arduino

Что происходит:

Пользователь нажимает «Читать данные»

Python ждёт ответ от Arduino (может занять секунды)

Интерфейс зависает — нельзя нажать другие кнопки

Через несколько секунд UI обновляется

**Решение.** threading для фоновых задач

Используйте отдельный поток для долгих операций:

```
python
1 # ✅ ХОРОШО: Не блокирует UI
2 import eel
3 import serial
4 import threading
5 import time
6
7 eel.init('www')
8
9 read_thread = None
10 stop_reading = False
11
12 @eel.expose
13 def start_reading(port):
14     """Запускает чтение в отдельном потоке"""
15     global read_thread, stop_reading
16
17     stop_reading = False
18     read_thread = threading.Thread(target=read_loop, args=(port,))
19     read_thread.daemon = True # Поток закроется вместе с приложением
20     read_thread.start()
21
22     return {"status": "success", "message": "Чтение запущено"}
23
```

Рис. 47. Используйте отдельный поток для долгих операций

```
24 @eel.expose
25 def stop_reading():
26     """Останавливает чтение"""
27     global stop_reading
28     stop_reading = True
29     return {"status": "success", "message": "Чтение остановлено"}
30
31 def read_loop(port):
32     """Фоновый цикл чтения данных"""
33     global stop_reading
34
35     try:
36         ser = serial.Serial(port, 9600, timeout=1)
37         time.sleep(1.5) # ждём инициализации Arduino
38
39         while not stop_reading and ser.is_open:
40             if ser.in_waiting > 0:
41                 line = ser.readline().decode('utf-8').strip()
42                 if line:
43                     # Вызываем JS для обновления UI
44                     eel.update_data(line)()
45                 time.sleep(0.1) # Небольшая пауза для стабильности
46
47     except Exception as e:
48         eel.show_error(str(e))()
49     finally:
50         if 'ser' in locals() and ser.is_open:
51             ser.close()
52
```

Рис. 48. Используйте отдельный поток для долгих операций (продолжение)

```
47     except Exception as e:
48         eel.show_error(str(e))()
49     finally:
50         if 'ser' in locals() and ser.is_open:
51             ser.close()
52
53 eel.start('index.html')
```

Рис. 49. Используйте отдельный поток для долгих операций (продолжение)

#### javascript

```
1 // www/app.js
2 eel.expose(function update_data(line) {
3     // добавляем данные в лог
4     const log = document.getElementById('data-log');
5     log.innerHTML += `<div>${new Date().toLocaleTimeString()}: ${line}</div>`;
6     log.scrollTop = log.scrollHeight; // Автопрокрутка вниз
7
8     // Обновляем график (если есть)
9     updateChart(line);
10 });
11
12 eel.expose(function show_error(message) {
13     document.getElementById('error').innerText = "Ошибка: " + message;
14     document.getElementById('error').style.display = 'block';
15 });
```

Рис. 50. Используйте отдельный поток для долгих операций (продолжение)

Преимущества подхода:

UI остаётся отзывчивым — можно нажимать кнопки во время чтения

Данные обновляются в реальном времени — каждая строка сразу отображается

Можно остановить чтение — кнопка «Стоп» работает мгновенно

Ошибки обрабатываются корректно — не ломают приложение

## 6.6. Режим эмуляции: тестирование без железа

Вспомните про DEBUG\_MODE из ваших проектов. Вот как это работает в контексте асинхронности:

```
python

1  # main.py
2  import eel
3  import random
4  import threading
5  import time
6
7  DEBUG_MODE = True # ← Переключите на False для реального устройства
8
9  eel.init('www')
10
11 connected = False
12 read_thread = None
13 stop_reading = False
14
15 @eel.expose
16 def get_ports():
17     """Возвращает список портов + эмулятор"""
18     import serial.tools.list_ports
19     ports = [port.device for port in serial.tools.list_ports.comports()]
20
21     if DEBUG_MODE:
22         ports.append("EMULATOR") # Добавляем опцию эмулятора
23
24     return ports
25
```

Рис. 51.

```
15 @eel.expose
16 def get_ports():
17     """Возвращает список портов + эмулятор"""
18     import serial.tools.list_ports
19     ports = [port.device for port in serial.tools.list_ports.comports()]
20
21     if DEBUG_MODE:
22         ports.append("EMULATOR") # Добавляем опцию эмулятора
23
24     return ports
25
26 @eel.expose
27 def connect(port):
28     """Подключение к порту или эмулятору"""
29     global connected, read_thread, stop_reading
30
31     if DEBUG_MODE and port == "EMULATOR":
32         connected = True
33         stop_reading = False
34         read_thread = threading.Thread(target=emulate_data)
35         read_thread.daemon = True
36         read_thread.start()
37         return {"status": "success", "message": "Подключено к эмулятору"}
38
39     # Реальное подключение (код из раздела 6.5)
40     # ...
41
```

Рис. 52. Режим эмуляции: тестирование без железа (продолжение)

```
41
42 def emulate_data():
43     """Генерирует правдоподобные данные датчика"""
44     global stop_reading, connected
45
46     base_temp = 25.0
47     while not stop_reading and connected:
48         # Добавляем случайное колебание (как у реального датчика)
49         noise = random.gauss(0, 0.5)
50         base_temp += random.gauss(0, 0.1)
51         base_temp = max(15, min(35, base_temp)) # Ограничиваем диапазон
52
53         value = round(base_temp + noise, 2)
54         eel.update_data(f"Датчик: {value}°C")
55         time.sleep(1) # 1 секунда между измерениями
```

Рис. 53. Режим эмуляции: тестирование без железа (продолжение)

**Совет.** Внедряйте режим эмуляции с первого дня разработки. Это сэкономит часы отладки, когда Arduino нет под рукой.

## 6.7. Обработка ошибок на обеих сторонах

Python → JavaScript

python

```
1 # main.py
2 @eel.expose
3 def safe_operation(data):
4     try:
5         # Опасная операция
6         result = process_data(data)
7         return {"status": "success", "result": result}
8     except Exception as e:
9         # Возвращаем ошибку как часть ответа
10        return {"status": "error", "message": str(e)}
```

*Рис. 54. Python → JavaScript*

javascript

```
1 // www/app.js
2 async function runOperation() {
3     const response = await eel.safe_operation(someData());
4
5     if (response.status === "success") {
6         console.log("Результат:", response.result);
7     } else {
8         // Показываем ошибку пользователю
9         showError(response.message);
10    }
11 }
```

*Рис. 55. Python → JavaScript (продолжение)*

JavaScript → Python

javascript

```
1 // www/app.js
2 eel.expose(function validateInput(value) {
3     if (!value || value === "") {
4         throw new Error("Поле не может быть пустым!");
5     }
6     if (isNaN(value)) {
7         throw new Error("Требуется числовое значение!");
8     }
9     return true;
10 });
```

*Рис. 56. JavaScript → Python*

```
python
1 # main.py
2 @eel.expose
3 def process_user_input(value):
4     try:
5         # Вызываем JS-валидацию
6         eel.validate_input(value)()
7         # Продолжаем обработку
8         return {"status": "success"}
9     except Exception as e:
10        return {"status": "error", "message": str(e)}
```

Рис. 57. JavaScript → Python (продолжение)

## 6.8. Частые ошибки и решения

| ПРОБЛЕМА              | ПРИЧИНА                          | РЕШЕНИЕ                                                   |
|-----------------------|----------------------------------|-----------------------------------------------------------|
| Функция не вызывается | Забыли <code>@eel.expose</code>  | Добавьте декоратор в Python                               |
| Две пары скобок?      | Синтаксис Eel                    | <code>eel.func()()</code> — это нормально!                |
| UI зависает           | Долгая операция в главном потоке | Используйте <code>threading</code>                        |
| JS-функция не найдена | Функция не глобальная            | Используйте <code>function</code> , не <code>const</code> |
| Данные не передаются  | Тип не сериализуется в JSON      | Преобразуйте в <code>dict</code> / <code>list</code>      |
| Порт не закрывается   | Поток не остановлен              | Используйте <code>stop_reading</code> флаг                |

Рис. 58.

Что мы узнали?

Вызов Python из JS: `await eel.function_name(args)()` — две пары скобок обязательны.

Вызов JS из Python: `eel.js_function()()` — функция должна быть глобальной.

Передача данных: Поддерживаются все JSON-сериализуемые типы (списки, словари).

Асинхронность: Долгие операции (COM-порт, файлы) — только в `threading`.

Режим эмуляции: Тестируйте без железа с `DEBUG_MODE`.

Обработка ошибок: Возвращайте статус `success/error` в словаре.

**Чек-лист.** Практическое задание

Создайте приложение «Монитор температуры»:

1.a. Кнопка «Подключиться» — выбирает порт или «EMULATOR».

1.b. Кнопка «Начать чтение» — запускает фоновый поток.

1.c. Кнопка «Стоп» — останавливает чтение.

1.d. Текстовое поле — показывает последние данные.

Реализуйте режим эмуляции:

2.a. Добавьте `DEBUG_MODE = True` в начало `main.py`.

2.b. Создайте функцию `emulate_data()` с генерацией случайной температуры.

2.c. Убедитесь, что приложение работает без Arduino.

Добавьте обработку ошибок:

3.a. Что будет, если порт занят?

3.b. Что будет, если пользователь нажмёт «Стоп» до «Начать»?

3.c. Реализуйте проверки и покажите понятные сообщения.

Поэкспериментируйте с данными:

## **Конец ознакомительного фрагмента.**

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.