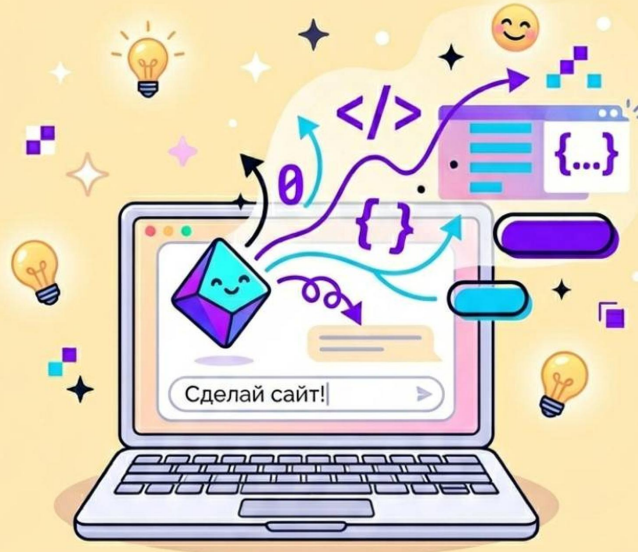
Программи-рование без программи-рования



Кейси

Кейси Кейси

Программирование без программирования

https://litres.ru/74014721
SelfPub; 2026

Аннотация

Мир изменился. Ещё вчера, чтобы написать сайт, приложение, бота или игру, нужно было годами учить языки программирования, разбираться в алгоритмах и помнить сотни синтаксических правил. Сегодня всё иначе. Появился вайб-кодинг — подход, при котором вы описываете идею словами, а код пишет искусственный интеллект.

Эта книга — не учебник по программированию. В ней нет сухой теории, зубрёжки терминов и академических глав «для общего развития». Это практическое руководство по созданию цифровых продуктов с помощью AI-инструментов: Cursor, Replit Agent, GitHub Copilot, Claude и других.

Вы узнаете:

Почему эпоха «кода ради кода» заканчивается и что это значит лично для вас.

Как правильно общаться с AI, чтобы он выдавал работающий код, а не бессмысленною простыню.

Как собрать свой первый проект за один вечер, даже если вы никогда не открывали редактор кода.

Где проходят границы вайб-кодинга: что AI делает хорошо, а что пока не умеет.

Содержание

Кейси
Программирование без программирования

ВВЕДЕНИЕ

Привет. Давай начистоту.

Если ты держишь в руках эту книгу (или скроллишь её на экране), скорее всего, тебе хотя бы раз в жизни казалось, что программирование - это какая-то закрытая секта. Со своим языком, ритуалами и дедовщиной. Где все вокруг обсуждают «фреймворки» и «микросервисы», а ты сидишь и думаешь: «Ребята, мне просто нужно, чтобы кнопка работала».

Я тебя понимаю. Меня зовут [Имя Автора], и я тоже когда-то думал, что код - это магия, доступная лишь избранным. А потом случилось странное: код перестал быть главным.

В 2024–2026 годах произошла тихая революция. Искусственный интеллект научился писать код лучше многих людей. Серьёзно. Он не устаёт, не путает скобки и не уходит в депрессию из-за бага в пятницу вечером. И это изменило правила игры.

Теперь главный навык — не умение писать код. А

умение объяснить, что ты хочешь создать.

Это и есть вайб-кодинг.

Название дурацкое, согласен. Пахнет стартап-тусовкой и смузи-культурой. Но суть верная. Ты ловишь вайб идеи, описываешь её словами, а AI превращает слова в работающий продукт. Ты не печатаешь код, ты дирижируешь процессом.

Эта книга - не учебник. Я не буду грузить тебя теорией, которую ты забудешь через день. Вместо этого мы вместе пройдём путь от «я никогда этого не делал» до «смотрите, я сделал это сам». С настоящими проектами, живыми примерами и ошибками, на которых я уже набил шишки.

Никакой воды. Никакого «программирование - это просто». Будет трудно. Будет непонятно. Иногда будет бесить. Но в конце ты создашь то, что раньше казалось невозможным.

Поехали.

Часть I. НОВАЯ РЕАЛЬНОСТЬ.

Глава 1. Джими Хендрикс, лифтёры и конец эпохи «чистого кода».

История, которая начинается в лифте

Представьте: Нью-Йорк, 1945 год. Вы заходите в солидный офисный небоскрёб. В лифте вас встречает человек в униформе. Он закрывает решётчатую дверь, поворачивает бронзовую ручку - и плавно везёт вас на четырнадцатый этаж. Этот человек - **лифтёр**. Уважаемая профессия. Стабильный заработок. Профсоюз.

Проходит двадцать лет и лифтёров в Нью-Йорке практически не осталось. Их убила не катастрофа и не забастовка. Их убила **кнопка**. Автоматизация. Человек нажал — лифт поехал. Профсоюзы бастовали, люди кричали о безопасности, но прогресс было не остановить.

А теперь вопрос на миллион долларов: **станут ли программисты новыми лифтёрами?**

Ещё пять лет назад этот вопрос звучал как шутка. Сегодня над ним уже никто не смеётся.

Потому что программирование как профессия стоит ров-

но перед той же автоматической дверью.

Человек, который играл не по нотам

Теперь другая история. Лондон, 1966 год. В студию заходит молодой гитарист. У него нет музыкального образования. Он не знает сольфеджио. Если перед ним положить ноты, он, скорее всего, скомкает их и выбросит в окно.

Но когда он берёт в руки гитару, происходит что-то странное. Звук, которого до него не существовало. Энергия, которая меняет музыку навсегда.

Этого парня звали **Джими Хендрикс**.

Он не знал теории. Но он знал главное, **какой звук хочет услышать**. И умел объяснить это гитаре.

Теперь следите за параллелью. До 2024 года программирование было похоже на классическую консерваторию. Хочешь создать что-то, учи ноты. Годами. Учи синтаксис, алгоритмы, структуры данных. Играй гаммы, пока пальцы не заболят.

А потом появился AI и оказалось, что можно как Хендрикс. Ты просто знаешь, что хочешь услышать. И объясняешь это инструменту. Не нотами - словами.

Это и есть вайб-кодинг.

Ты не программист в классическом смысле. Ты человек с идеей. А код теперь пишет тот, кто делает это быстрее и без ошибок.

Почему «печатать код» перестало быть главным навыком

Давайте честно. Последние сорок лет программирование выглядело так:

1. Ты изучаешь язык.
2. Ты запоминаешь правила.
3. Ты печатаешь символы в редакторе.
4. Ты молишься, чтобы оно запустилось.
5. Оно не запускается.
6. Ты ищешь пропущенную запятую три часа.

Весь фокус был на шаге номер три - **напечатать правильные символы в правильном порядке**. Именно этому учили в вузах, на курсах, в книжках. Именно это считалось «настоящим программированием».

Но теперь шаг номер три умеет делать машина.

Это как если бы вы учились набирать текст на печатной машинке, а вам дали голосовой ввод. Да, навык печати - полезный. Но он больше не является входным билетом в профессию.

Главным становится другое:

- Понять, что именно нужно создать.

- Разбить идею на части, которые можно объяснить.

- Проверить результат и сказать, что не так.

- Повторить, пока не получится.

Это не про код. Это про мышление. Про умение видеть продукт целиком. Про вкус, в конце концов. Как у Хендрикса - он не ноты знал, он звук чувствовал.

История одного школьника

У меня есть знакомый. Назовём его Димой. Ему пятнадцать лет, он живёт в Твери и ненавидит математику.

Полгода назад Дима захотел создать игру. Не сложную - простую аркаду, где человечек бегает по платформам и собирает монетки. В школе ему сказали: «Сначала выучи Pascal, потом основы алгоритмов, потом может быть, через пару лет напишешь что-то простое».

Дима плюнул. Он открыл Claude, написал: «Сделай мне простую браузерную игру, где персонаж бегает и собирает предметы. Хочу управлять стрелками. Персонаж - синий квадрат. Предметы - жёлтые круги. Веди счёт».

Через три часа у него была работающая игра. Со счётчиком, экраном победы и даже музыкой, которую AI предложил добавить сам.

Дима не знает, что такое «функция» и «цикл». Он не сможет написать эту игру с нуля. Но он сделал то, что хотел. И главное, он понял, **как объяснять машине свои желания**.

Через месяц он сделал телеграм-бота для школьного чата.

Ещё через месяц сайт для маминого маленького бизнеса.

Дима не стал программистом. Он стал **вайб-кодером**.

И вот что важно: его результаты не хуже, чем у студента второго курса технического вуза. Иногда лучше. Потому что AI не забывает точки с запятой.

Так что, программисты больше не нужны?

Нет. Это не так.

Но **изменилась точка входа**.

Раньше путь выглядел так: *Учебник синтаксис задачи пет-проект работа.*

Теперь путь выглядит иначе: *Идея диалог с AI прототип итерации продукт.*

Это не значит, что «настоящие программисты» исчезнут. Врачи не исчезли, хотя появились медицинские справочники и онлайн-диагностика. Но появилось что-то новое: человек, который может помочь себе сам в простых случаях.

Вайб-кодинг - это скорая помощь для идей. Ты можешь создать MVP, проверить гипотезу, запилить лендинг или бота для внутренних нужд, не нанимая команду и не тратя месяцы на обучение.

А если проект взлетит, вот тогда придёт «настоящий программист». Который перепишет это дело на нормальной архитектуре, чтобы оно не развалилось при масштабировании. Но он придёт не на пустое место. Он придёт на работающий

прототип, который ты создал за выходные.

Вайб-момент

В 1980-х говорили: «Компьютер - это сложно, тебе нужен специальный человек».

В 2000-х говорили: «Сайт - это дорого, тебе нужна студия».

В 2010-х говорили: «Приложение - это долго, тебе нужна команда».

В 2026 году всё это можно сказать искусственному интеллекту. И он ответит: «Окей, сейчас сделаю».

Единственное, что теперь нужно - **знать, чего ты хочешь**.

Что мы выяснили в этой главе

1. Вайб-кодинг - это не хайп и не мошенничество. Это следующий этап автоматизации, который мы уже проходили десятки раз в истории. Лифтёры, машинистки, телефонистки - все они уступили место технологии, которая сделала процесс проще и доступнее.

2. Программирование перестало быть навыком набора символов и стало навыком постановки задачи. Твоя ценность не в умении помнить синтаксис, а в умении

видеть, что должно получиться, и объяснять это машине.

3. Ты не обязан становиться программистом, чтобы создавать цифровые продукты. Как не обязан быть механиком, чтобы водить машину, или поваром, чтобы приготовить ужин. Инструменты изменились, порог входа упал. Грех этим не воспользоваться.

Задание для самостоятельной практики

Сегодня простое, но важное задание. Без кода.

1. Подумайте о какой-нибудь маленькой цифровой штуке, которую вам хотелось бы иметь. Не «социальная сеть» и не «убийца YouTube». Что-то микроскопическое. Сайт-визитка. Бот, который по утрам присылает погоду. Игра в угадайку. Калькулятор привычек.

2. Запишите идею на листе бумаги или в заметках. Не думайте о реализации. Просто опишите: **что эта штука делает, как она выглядит, кто ей пользуется и зачем.**

3. Отложите запись до следующей главы. Она нам пригодится.

Глава 2. Кто такой вайб-кодер и чем он отличается от «настоящего программиста».

Пять мифов, которые нам мешают

Прежде чем разбираться, кто такой вайб-кодер, давайте вытащим из головы мусор, который туда заботливо напихали за последние двадцать лет. Это важно, потому что если не вытащить, дальше книга будет вызывать у вас внутреннее сопротивление. Мозг будет твердить: «Ну нет, программирование - это сложно, меня обманывают, так не бывает».

Давайте разберём пять мифов. Спокойно, без истерики.

Миф 1. «Чтобы создать программу, нужно знать языки программирования»

Этот миф возник потому, что раньше по-другому было нельзя. Компьютер понимает код, значит, ты должен говорить на коде. Логично.

Но теперь между вами и компьютером появился переводчик. Он говорит на Python, JavaScript, C++ и ещё полусотне языков. А вы говорите на русском. Или на английском. Или на смеси, ему без разницы.

Сравнение: вам не нужно знать китайский, чтобы заказать товар на AliExpress. Сайт переведёт описание, а продавец сам как-нибудь разберётся. AI - это такой же слой перевода между вами и машиной. Вы даёте задачу словами - он превращает её в код.

Знание языков программирования теперь **продвинутый навык**, а не входной билет. Полезный? Безусловно. Обязательный для старта? Уже нет.

Миф 2. «Программисты мыслят как-то особенно, а обычным людям это не дано»

Есть такое понятие «инженерное мышление». О нём написаны горы книг, и почти все они создают ощущение, что программисты это отдельный биологический вид. Типа эльфов: живут долго, видят в темноте, мыслят алгоритмами.

Чушь.

Программисты - обычные люди, которые просто потратили много часов на решение определённого типа задач. Их мозг натренирован, как мышцы у бегуна. Но это не значит, что вы не можете пробежать стометровку.

Более того: вайб-кодинг требует другого типа мышления. Не алгоритмического, а **продУктового**. Вы должны понимать не «как отсортировать массив», а «что нужно пользователю на экране». А этому в институтах как раз не учат или учат плохо.

Так что если вы гуманитарий, предприниматель, дизайнер, маркетолог, у вас может быть **форА** перед класси-

ческим программистом. Вы уже знаете, что нужно людям. Осталось научиться объяснять это машине.

Миф 3. «AI пишет говнокод, которым невозможно пользоваться»

Это мой любимый миф. Потому что он одновременно и правда, и ложь.

Правда: AI иногда пишет неоптимальный код. Может сгенерировать лишнее, запутанное, некрасивое решение. Код, который работает, но который профессиональный разработчик переписал бы элегантнее.

Ложь: это имеет значение.

Смотрите. Если вы делаете лендинг для бабушкиного пирожкового бизнеса, вам плевать на «чистоту архитектуры». Вам нужно, чтобы страница загрузилась и клиент оставил заявку. Всё. Работает? Работает. Бабушка счастлива? Счастлива. Значит, код достаточно хорош.

Профессиональные программисты часто страдают «синдромом идеального кода». Они будут полировать решение неделями. А ваш лендинг уже работает, приносит заявки и окупается. Пока они допиливают архитектуру, вы допиливаете бизнес.

Оптимизация нужна, когда у вас сто тысяч пользователей. А когда их десять, годится любой рабочий код. Не дайте лучшему стать врагом хорошего.

Миф 4. «Если я не понимаю, как оно работает, я не смогу это контролировать»

О, это глубокая экзистенциальная тревога. Потеря контроля. Мы все через неё проходим.

Но давайте проверим: вы понимаете, как работает двигатель вашей машины? А холодильник? А алгоритм рекомендаций YouTube? Вы пользуетесь всем этим каждый день, не имея ни малейшего понятия о внутреннем устройстве.

Контроль в вайб-кодинге - это не понимание каждой строчки. Это понимание **поведения**. Ваша задача: проверить, что программа делает то, что вы просили. Не как она это делает, а что она в итоге выдаёт.

Если бот отвечает на сообщения - он работает. Если сайт открывается - он работает. Если игра запускается - она работает. Вы контролируете результат. Метод достижения результата вы делегировали AI. Как делегируете водителю автобуса задачу довезти вас до метро, вам не нужно знать, как переключать передачи на МАЗе.

Миф 5. «Это всё баловство, скоро тренд пройдёт»

Знаете, что говорили про интернет в 1995 году? «Игрушка для гиков, не серьёзно».

Что говорили про iPhone в 2007? «Дорогая игрушка, кнопок нет, бизнесу не нужно».

Что говорили про облачные технологии? «Данные должны быть на своём железе, это вопрос безопасности».

Вайб-кодинг - не тренд. Это прямое следствие развития языковых моделей. Они не станут глупее. Они станут только умнее, быстрее и дешевле. Через год AI будет писать код

ещё лучше. Через три, возможно, заменит целые категории задач, которые сегодня делают джуниоры.

Окно возможностей открыто прямо сейчас. Не потому что это модно. А потому что это навсегда.

Три типа людей в новом мире разработки

Итак, мифы мы развеяли. Теперь давайте нарисуем карту местности. Кто вообще населяет мир создания цифровых продуктов в 2026 году?

Я вижу три большие группы. Не профессии, скорее, роли. Способы мышления.

Тип первый. Архитектор

Это человек, который видит продукт целиком. Он не пишет код - он проектирует систему. Он понимает пользователя, рынок, задачу, которую продукт решает.

Раньше архитекторами становились программисты, которые прошли путь от джуна до сеньора и научились видеть лес за деревьями. Но теперь, и это важно, архитектором может стать человек вообще без технического бэкграунда. Если у него есть видение продукта и умение разбивать большие задачи на маленькие.

Архитектор спрашивает:

- Какую проблему пользователя мы решаем?
- Какой путь пользователь пройдёт от входа до результата?
- Из каких блоков будет состоять продукт?

Архитектор не обязан знать, как именно AI реализует базу данных. Он должен знать, что база данных тут нужна.

Тип второй. Код-мастер

Это классический программист. Человек, который глубоко понимает технологии. Может написать сложный алгоритм, оптимизировать запрос, построить масштабируемую архитектуру.

Код-мастеры никуда не денутся. Они будут нужны всегда - для сложных, высоконагруженных, нестандартных систем. Но их роль меняется. Раньше они делали всё - от кнопки до сервера. Теперь рутину забирает AI, а код-мастер занимается тем, что AI пока не умеет: принимает архитектурные решения, ищет нестандартные ходы, лезет в дебри, куда у модели не хватает контекстного окна.

Код-мастер спрашивает:

- Как сделать это оптимально?
- Не упадёт ли это под нагрузкой?
- Какую технологию выбрать для нестандартной задачи?

Код-мастер и вайб-кодер не враги. Это разные роли, которые могут работать в команде. И часто в одном человеке.

Тип третий. Вайб-кодер

Это вы. Или вы через несколько глав.

Вайб-кодер - это человек, который создаёт цифровые продукты через диалог с AI. Он не пишет код руками. Он описывает задачу, получает решение, тестирует его, даёт обратную связь, итерирует.

Что нужно вайб-кодеру:

- Умение чётко формулировать, что должно получиться.
- Базовое понимание логики (если A, то B).
- Готовность к экспериментам («не получилось - попробую иначе»).
- Насмотренность: знать, какие продукты вообще бывают и что можно попросить у AI.

Что вайб-кодеру не нужно:

- Знать синтаксис языков программирования.
- Понимать, как работает сервер.
- Уметь настраивать базы данных вручную.
- Помнить сотни команд терминала.

Вайб-кодер находится на другом уровне абстракции. Он управляет не кодом, он управляет **намерением**.

Почему «лень» - это суперсила

Есть старая шутка: «Хороший программист - ленивый программист. Потому что он потратит час на автоматизацию задачи, которую можно сделать руками за пять минут, зато потом она будет делаться сама».

В этой шутке глубокая мудрость.

Лень (в правильном смысле) - это нежелание делать тупую повторяющуюся работу. Это инстинктивное стремление к эффективности. Вайб-кодер - это человек, чья лень достигла просветления. Он не просто не хочет писать код, он не

хочет делать ничего, что можно делегировать.

И это правильно.

Потому что ваше время и внимание - самый ценный ресурс. Если AI может сделать что-то за вас - позвольте ему. Не из слабости. Из стратегии.

Пока классический программист пишет сотую функцию авторизации в своей жизни, вайб-кодер описывает её одной фразой и думает над следующей фичей.

Лень - это фильтр. Если задача рутинная - делегируй. Если творческая, уникальная, требует вкуса - делай сам. Вайб-кодинг позволяет вам сфокусироваться на том, что действительно требует человека.

Вайб-момент

Однажды мой знакомый код-мастер с пятнадцатилетним стажем сказал мне:

«Знаешь, что самое обидное? Я пятнадцать лет учился писать код. А теперь ты со своим AI делаешь прототип быстрее меня. Не лучше — но быстрее. И пока я дописываю идеальную архитектуру, твой прототип уже прошёл три итерации с пользователями и нашёл правильный ответ».

Он был не зол. Он был задумчив.

А через месяц сам начал использовать AI-помощников. Потому что лень, как я уже говорил, - это суперсила.

Что мы выяснили в этой главе

1. Пять главных страхов о вайб-кодинге - мифы. Они держатся на устаревшей картине мира, где код был единственным способом общения с компьютером. Теперь появился переводчик, и правила изменились.

2. В новом мире есть три роли: Архитектор, Код-мастер и Вайб-кодер. Они не конкурируют - они дополняют друг друга. Вы можете быть кем-то одним или совмещать. Но начинать проще всего с роли Вайб-кодера.

3. Не нужно понимать код, чтобы контролировать результат. Вы контролируете поведение продукта. Если продукт делает то, что вы хотели, - вы управляете процессом. Остальное - детали реализации, которые можно делегировать.

Задание для самостоятельной практики

Найдите в интернете три любых цифровых продукта, которые вам нравятся. Можно сайты, можно приложения, можно ботов. Запишите для каждого один ответ на вопрос:

«Если бы я объяснял создание этого продукта AI, с чего бы я начал? Как бы я описал его в трёх предложениях?»

Не думайте пока о том, «реально ли это сделать». Просто попробуйте описать. Это первый шаг к мышлению архитектора.

Глава 3. Инструментарий: ваш новый лучший друг.

Эпоха «голых рук» закончилась

Раньше, чтобы начать программировать, нужно было пройти ритуал посвящения. Сначала выяснить, какой язык учить. Потом скачать IDE (что такое IDE? ага, уже непонятно). Потом настроить окружение. Потом прочитать три гайда по настройке окружения, потому что первые два устарели. Потом установить расширения, плагины, пакетные менеджеры... К моменту, когда вы были готовы написать первую строчку кода, проходила неделя и заканчивался энтузиазм.

Хорошие новости: этот этап мы перепрыгиваем.

Вайб-кодеру не нужен настроенный до блеска IDE с двадцатью плагинами. Не нужно знать, какой менеджер пакетов лучше. Не нужно вообще понимать, что такое «менеджер пакетов».

Нужно три вещи:

1. Инструмент, в котором вы будете общаться с AI.
2. Браузер.
3. Интернет.

Всё. Остальное опционально и будет наращиваться по ме-

ре необходимости.

Но чтобы вы не ткнули пальцем в первый попавшийся сервис и не разочаровались, давайте спокойно разложим, что есть что, какой инструмент для чего заточен и с чего начать именно вам.

Какие бывают инструменты: три категории

Современные AI-инструменты для создания кода я бы разделил на три большие группы. Представьте, что вы собираетесь приготовить ужин.

Категория 1. Текстовые AI-помощники (ваш «мозговой штурмовик»)

Это чаты: Claude, ChatGPT, Gemini. Вы открываете вкладку браузера, пишете: «Сделай мне то-то» и получаете код в ответ. Код нужно скопировать, вставить куда-то, запустить.

Плюсы: не требует установки, можно использовать с телефона, отлично для обсуждения идей и генерации отдельных кусков кода. Минусы: не видит проект целиком, код надо таскать туда-сюда руками.

Это как спросить у друга-повара рецепт по телефону. Готовить всё равно вам.

Категория 2. AI-IDE (ваша «кухня со встроенным шеф-поваром»)

Это среда разработки с AI внутри. Cursor - главный представитель. Вы открываете программу, она видит все файлы

проекта, и AI понимает контекст. Вы говорите: «Добавь на сайт тёмную тему» и AI меняет код в нужных файлах сам. Не надо ничего копировать.

Плюсы: глубокое понимание проекта, AI работает прямо в коде, всё в одном окне. Минусы: надо установить программу, выглядит как «настоящая среда разработки» (может испугать новичка).

Это как готовить на профессиональной кухне, где шеф стоит рядом и помогает.

Категория 3. No-code / Low-code платформы (ваш «ресторан с доставкой»)

Это сервисы вроде Bolt.new, v0, Lovable, Replit. Вы описываете идею текстом, а сервис сам собирает приложение целиком и показывает результат в браузере. Можно ничего не устанавливать.

Плюсы: минимальный порог входа, быстрое прототипирование, результат видно сразу. Минусы: меньше контроля, сложнее кастомизировать, часто требуют платной подписки на серьёзных проектах .

Это как заказать готовое блюдо. Быстро, вкусно, но рецепт не ваш и ингредиенты не вы выбирали.

Знакомство с героями: кто есть кто

Давайте пройдёмся по конкретным инструментам, которые вам встретятся. Не как в обзоре «10 лучших сервисов

для вайб-кодинга», а по-человечески: что это, зачем нужно и нужно ли вообще.

Cursor - ваш основной инструмент

Cursor - это редактор кода с AI внутри. Внешне почти неотличим от VS Code (популярнейшего редактора для программистов), но со встроенным интеллектом . Главное, что нужно знать: вы пишете в чат, ЧТО хотите сделать — и Cursor меняет код. Он видит весь проект, понимает структуру файлов и может работать сразу в нескольких местах .

Возможностей много, но новичку достаточно запомнить три комбинации клавиш:

Ctrl+L (на Mac — Cmd+L) - открыть чат с AI. Спрашивайте что угодно.

Ctrl+K (Cmd+K) - изменить выделенный кусок кода. Выделили написали, что поменять готово .

Ctrl+I (Cmd+I) - «режим агента». Для больших задач: «Создай страницу с формой обратной связи» .

Бесплатной версии хватит для старта. Платная ($20/мес) открывает больше возможностей, но в первых главах этой книги она не понадобится .

Совет: Если Cursor кажется сложным - не пугайтесь. В следующей главе мы пошагово запустим ваш первый проект. Это как первый раз за руль: страшно, пока не попробуешь.

Claude - «думающий» помощник

Claude от Anthropic - это AI, который славится умением работать с большими объёмами кода и сложными задачами

. Он хорош для ситуаций, когда нужно не просто сгенерировать код, а разобраться в проблеме.

Когда использовать: вы запутались в ошибках, Cursor не справляется, нужно спокойно обсудить архитектуру. Claude «думает» аккуратнее многих конкурентов и неплохо объясняет свои решения .

Есть бесплатная версия на claude.ai. Оплата зарубежной картой, что сейчас может быть проблемой, но для первых шагов бесплатного лимита достаточно.

ChatGPT - универсальный солдат

ChatGPT от OpenAI - инструмент, с которого многие начинали. Хорош для быстрых вопросов, генерации отдельных функций, объяснения ошибок. Уступает Claude в глубине работы с большим проектом, но берёт доступностью и скоростью.

Бесплатная версия доступна без всяких карт. Для вайбкодинга подходит на старте, но по мере усложнения проектов вы, скорее всего, перейдёте на специализированные инструменты.

Replit - облачная песочница

Replit - это браузерная среда, где можно писать код, запускать его и тут же публиковать результат . Внутри есть AI-помощник. Главный плюс: не надо ничего устанавливать. Открыли сайт написали промпт получили работающее приложение .

Хорош для: быстрых прототипов, обучения, работы с ком-

пьютера, на который нельзя устанавливать программы (например, рабочего ноутбука с ограничениями). Минус: платный на сколько-нибудь серьёзном использовании, от $18/мес .

Bolt.new и v0 - «сделай мне красиво»

Эти инструменты заточены на быструю генерацию веб-интерфейсов. Описываете, как должен выглядеть сайт или компонент - получаете готовый код. Bolt.new умеет создавать полноценные приложения с фронтендом и бэкендом прямо в браузере . v0 (от Vercel) специализируется на React-компонентах .

Хороши для: дизайнеров, которым нужно быстро превратить идею в работающий интерфейс. Минусы: ограниченный контроль над результатом, платные тарифы от $20-25/мес .

Что выбрать прямо сейчас?

Если голова пошла кругом от выбора - вот простой алгоритм. Никакой аналитики, чистая практика.

Шаг 1. Заведите аккаунт в Claude или ChatGPT (бесплатно). Это ваш «советчик» на случай, если что-то пойдёт не так.

Шаг 2. Скачайте и установите Cursor с сайта cursor.com . Это ваш основной инструмент для всей книги.

Шаг 3. Зарегистрируйтесь в Replit (бесплатного тарифа хватит) . Это запасной вариант для случаев, когда Cursor по-

кажется слишком сложным, или захочется быстро что-то задеплоить без лишних движений.

Всё. Три инструмента, один час на настройку - и вы вооружены для любой задачи из этой книги.

Зачем нужен «запасной» инструмент? Потому что AI иногда тупит. Бывает, Cursor зацикливается на ошибке и не может вылезти. В таких случаях вы открываете Claude, скармливаете ему проблему и просите подсказать решение. Два AI лучше, чем один, как два мозга лучше, чем один.

Настройка Cursor: без паники

Установка Cursor предельно проста. Заходите на cursor.com, жмёте Download, запускаете установщик. Всё как с любой программой .

При первом запуске Cursor предложит войти в аккаунт. Заведите бесплатный через Google или GitHub. Это нужно, чтобы AI заработал .

Если вы раньше пользовались VS Code, Cursor предложит перенести настройки. Если не пользовались просто пропустите этот шаг.

На этом настройка завершена. В следующей главе мы создадим первый проект, и вы увидите, как это работает на практике. Пока что достаточно просто запустить программу и убедиться, что она открылась.

Вайб-момент

На заре автомобилестроения водителю нужно было самому заводить мотор рукояткой, переключать скорости без синхронизаторов и быть немного механиком. Сегодня вы садитесь в машину и едете. Автоматическая коробка передач, кнопка Start, система помощи при парковке.

С инструментами для вайб-кодинга та же история. Раньше «вести машину» означало копаться в моторе. Теперь, просто знать, куда вы хотите приехать.

Cursor, Claude, Replit - это коробка-автомат. Не надо стыдиться, что вы не умеете переключать передачи вручную. Надо радоваться, что доедете быстрее и с комфортом.

Что мы выяснили в этой главе

1. Инструментов много, но начинать надо с трёх: текстовый AI-помощник (Claude или ChatGPT), AI-редактор кода (Cursor) и облачная песочница (Replit). Этого хватит для 95% задач новичка.

2. Не надо настраивать сложное окружение. Современный вайб-кодинг начинается с открытия сайта или установки одной программы. Старая боль «неделя настройки пять минут кода» ушла в прошлое .

3. Cursor - ваш главный инструмент. Освойте три горячие клавиши (чат, онлайн-редактирование, агент) — и сможете создавать проекты, не прикасаясь к коду руками .

Задание для самостоятельной практики

1. Скачайте и установите Cursor с cursor.com. Запустите, войдите в бесплатный аккаунт. Убедитесь, что программа открылась .

2. Зарегистрируйтесь на claude.ai или chatgpt.com (бесплатный аккаунт). Задайте любой вопрос про код, просто чтобы попробовать. Например: «Объясни, как работает цикл for, как будто мне десять лет».

3. Откройте Replit.com, создайте бесплатный аккаунт. Нажмите Create Repl, выберите любой шаблон и просто посмотрите, как выглядит среда .

4. Цель не создать шедевр, а пощупать инструменты. Чтобы в следующей главе, когда мы пойдём в практику, вы не отвлекались на технические мелочи.

Часть II. УЧИМСЯ ГОВОРИТЬ С AI.

Глава 4. Промпт-инжиниринг для простых смертных.

Почему «сделай мне красиво» не работает

Вайб-кодинг начинается с текста. Вы пишете слова, AI превращает их в код. Звучит магически. Но есть нюанс.

Помните старую сказку про золотую рыбку? Старуха просила корыто - получила. Просила избу - получила. Просила стать царицей - стала. Но когда попросила «быть владычицей морскою», всё рухнуло обратно к разбитому корыту.

Почему? Потому что запрос был **неконкретным и несоразмерным**.

С AI та же история. Если вы пишете: «Сделай мне сайт для бизнеса», он сделает. Какой-то. Возможно, даже красивый. Но почти наверняка не тот, который вы представляли.

И вот вы сидите, смотрите на результат и думаете: «Ну нет, вайб-кодинг - это ерунда, AI ничего не умеет».

AI умеет. Просто вы говорите с ним на разных языках. Вы на языке образов и желаний. Он на языке инструкций.

Промпт-инжиниринг - это искусство перевода с человеческого на «понятный AI».

И нет, это не rocket science. Сейчас разложим.

Формула хорошего промпта

Представьте, что вы нанимаете фрилансера. Допустим, веб-дизайнера. Вы говорите: «Сделай сайт». Он спрашивает: «Какой?» Вы: «Ну, красивый». Он: «О чём сайт? Какие страницы? Какая целевая аудитория? Какой стиль?»

Вы не сможете ответить и получите либо отказ, либо сайт-заглушку с Lorem Ipsum и стандартным шрифтом.

С AI то же самое. Но есть разница: AI не переспрашивает. Он заполняет пробелы сам. И часто заполняет их ерундой.

Формула хорошего промпта:

Контекст + Роль + Задача + Формат + Ограничения

Разберём по косточкам.

1. Контекст: «Кто я и что происходит»

AI не знает ничего о вас и вашем проекте. Если вы не скажете ему — он додумает. А додумывает он из среднестатистических данных интернета, что часто мимо.

Сравните:

«Сделай калькулятор»

и

«Я владелец пекарни, мне нужен калькулятор для расчёта стоимости тортов. Клиент выбирает размер, начинку и оформление, калькулятор показывает итоговую цену»

Почувствовали разницу? Во втором случае AI понимает не просто «что делать», а «зачем и для кого». Это критиче-

ски меняет результат.

Что сообщить в контексте:

- Кто вы (новичок, владелец бизнеса, дизайнер).
- Что за проект (сайт-портфолио, бот для сотрудников, игра для ребёнка).
- Кто конечный пользователь.

2. Роль: «Кем ты должен быть»

Это опциональный, но мощный приём. Вы можете сказать AI: «Ты - опытный Python-разработчик» или «Ты - дизайнер интерфейсов». AI подстроит стиль ответа и уровень сложности.

Для вайб-кодинга особенно хорошо работает: *«Ты - терпеливый преподаватель, который объясняет код новичкам»*.

Это снижает шанс, что AI выдаст простыню сложного кода без комментариев.

3. Задача: «Что конкретно сделать»

Главное правило: одна задача - один промпт.

Не «Сделай сайт с каталогом, корзиной и чатом». Разбейте:

Сначала: «Создай главную страницу с шапкой и пятью разделами».

Потом: «Добавь в шапку логотип и меню из трёх пунктов».

Потом: «Сделай раздел "О компании" с текстом и фотографией».

AI, как и человек, лучше справляется с маленькими чёткими задачами, чем с гигантским расплывчатым заданием.

4. Формат: «В каком виде выдать ответ»

Не стесняйтесь указывать формат. AI может выдать просто код, может код с пояснениями, может пошаговую инструкцию.

Примеры:

«Выдай HTML-код целиком, без пояснений»

«Объясни каждую строчку кода комментарием»

«Разбей ответ на шаги: сначала создай структуру, потом стили, потом логику»

Без указания формата AI выберет сам. Иногда угадает, иногда нет.

5. Ограничения: «Что важно и чего нельзя»

Это простая, но часто забываемая часть. AI нужно сказать, что для вас приоритетно, а чего вы точно не хотите.

Примеры ограничений:

«Используй только чистый HTML и CSS, без фреймворков»

«Кнопка должна быть зелёной»

«Сайт должен открываться на мобильных устройствах»

«Не используй внешние библиотеки»

«Код должен работать в браузере без сервера»

Ограничения сужают поле фантазии AI и приближают результат к вашей картинке в голове.

Собираем всё вместе: примеры

Давайте посмотрим, как формула работает на практике. Возьмём одну и ту же задачу и прокачаем её от «новичкового» промпта до «профессионального».

Уровень «Новичок» (плохо):

«Сделай таймер»

Результат: AI сделает какой-то таймер. Возможно, консольный. Возможно, на Python. Возможно, без кнопок. Что вы хотели - неизвестно.

Уровень «Средний» (уже неплохо):

«Сделай таймер обратного отсчёта на веб-странице. Чтобы были кнопки Старт, Пауза и Сброс. Таймер показывает минуты и секунды»

Результат: рабочий таймер в браузере. Но дизайн стандартный, цвет серый, шрифт системный. Работает, но скучно.

Уровень «Профи» (отлично):

«Я ведущий онлайн-тренировок. Мне нужен таймер обратного отсчёта для сайта — буду проецировать на экран во время занятий.

Контекст: тренировки групповые, люди смотрят на экран с расстояния 3-5 метров.

Задача: Создай веб-страницу с таймером обратного отсчёта. Таймер показывает минуты и секунды крупными цифра-

ми. Кнопки: Старт, Пауза, Сброс. При достижении нуля таймер меняет цвет фона на красный и издаёт звуковой сигнал (без внешних файлов, через Web Audio API).

Формат: Один HTML-файл, всё в одном и разметка, и стили, и скрипт.

Ограничения: Цифры должны быть видны с пяти метров (крупный шрифт, контрастные цвета). Кнопки крупные, чтобы нажимать на бегу. По умолчанию 60 секунд отсчёта. Не используй библиотеки, только нативные технологии.»

Конец ознакомительного фрагмента.