# ВАЙБКОДИК

## ДЛЯ ГУМАНИТАРИЕВ И НЕ ТОЛЬКО



<МАКЛЮК МАКСИМ />

Максим Маклюк

# ВАЙБКОДИК для гуманитариев и не только

«Автор»
2026

**Маклюк М. Р.**

ВАЙБКОДИК для гуманитариев и не только / М. Р. Маклюк — «Автор», 2026

Я потратил несколько месяцев, чтобы ты смог разобраться в этом за неделю.Хочешь сделать свою игру? Приложение? Или сайт? Понадобится один промт! И одна книга.Здесь нет заумных лекций. Только практика: как общаться с ИИ, как собирать приложения, как находить ошибки и как публиковать свои проекты в интернете.Ты не будешь учиться программировать годами. Ты научишься создавать проекты с помощью искусственного интеллекта. И на это уйдет всего одна неделя. Вперед!

© Маклюк М. Р., 2026
© Автор, 2026

# Содержание

1.1. Как работает браузер и что такое «код» 6
1.2. Базовый HTML: семантика и структура страницы 7
1.3. Базовый CSS: Flexbox, Grid и адаптивность 10
1.4. Основы JavaScript: переменные, функции, массивы и объекты 13
1.5. Что такое DOM и как скрипты меняют интерфейс 15
1.6. Терминал и файловая система: минимум для работы 17
2.1. Среда IDE (VS Code) и ИИ-ассистенты: куда вставлять код 19
2.2. Введение в Git и GitHub: контроль версий с первого дня 21
2.3. Как формулировать техническое задание для ИИ (TDD-подход) 22
2.4. Итеративный подход: дробление задач на мелкие шаги 24
Конец ознакомительного фрагмента. 25

# Максим Маклюк
# ВАЙБКОДИК для гуманитариев и не только

**ВАЙБОДИК для гуманитариев и не только**

**Привет, читатель!**

Хочешь написать свою первую игру, программу или другой интересный проект? Теперь это возможно. Вайбкодинг тебе поможет.

Раньше для создания сайтов и приложений требовались годы изучения сложного кода. Сегодня всё изменилось. Ты придумываешь идею и ставишь задачи, а искусственный интеллект берет на себя всю техническую работу.

Эта книга — не скучный учебник. Здесь не будет сотен страниц сложной теории. Наша цель — дать тебе понимание, как всё устроено: от работы браузера до публикации готового проекта в интернете.

Что тебя ждет в цифрах:

10 минут в день — столько в среднем занимает чтение одной главы. Можно читать в метро или за утренним кофе.

3-4 часа — столько времени уйдет на изучение всей книги, если не делать больших перерывов.

От 0 до 100 — так вырастет твой уровень компетенций. Ты пройдешь путь от «не знаю, что такое код» до «могу спроектировать, собрать и задеплоить приложение».

Кем ты станешь к концу книги?

Ты не станешь классическим «программистом», который годами зубрит алгоритмы. Ты станешь создателем. Ты будешь тем человеком в мире программистов, который понимает, как работает вся система, умеет ставить задачи ИИ и превращать сырые идеи в работающие продукты.

Вайбкодинг — это про творчество и практику. Вперед!

# 1.1. Как работает браузер и что такое «код»

Прежде чем просить искусственный интеллект написать программу, нужно понять, где именно эта программа будет жить и как она работает. Начнем с самого главного.

**Что такое «код» на самом деле**

Для гуманитария код часто звучит как заклинание. На самом деле код — это просто **язык инструкций**. Это способ объяснить компьютеру, что ты хочешь сделать, шаг за шагом. Компьютер не умеет читать мысли, он понимает только строгие команды. Вайбкодинг хорош тем, что тебе не обязательно учить этот язык зубрить: искусственный интеллект возьмет перевод с человеческого на компьютерный на себя.

**Клиент-серверная архитектура (как работает интернет)**

Интернет устроен как поход в ресторан.

**Клиент** — это ты, посетитель. Твой компьютер или телефон. У тебя есть устройство (браузер), которым ты пользуешься.

**Сервер** — это кухня ресторана. Это мощный компьютер, который хранит файлы сайта (картинки, тексты, код) и постоянно включен.

**Браузер** — это официант. Ты говоришь официанту, что хочешь (вводишь адрес сайта). Официант бежит на кухню (отправляет запрос на сервер), забирает готовое блюдо (получает файлы сайта) и приносит тебе на стол, красиво сервируя (отрисовывает страницу на экране).

То есть, когда ты открываешь сайт, твой браузер стучится на сервер и просит: «Дай мне файлы этой страницы». Сервер отдает их браузеру, а браузер собирает из них картинку, которую ты видишь.

**Что именно присылает сервер**

Сервер отдает браузеру набор файлов. Главные из них — это HTML, CSS и JavaScript.

**HTML** — это скелет. Он говорит браузеру: «Здесь будет заголовок, здесь текст, а здесь картинка».

**CSS** — это внешний вид. Он говорит: «Сделай заголовок красным, а текст — по центру».

**JavaScript** — это мышцы. Это код, который оживляет страницу. Он говорит: «Когда пользователь нажмет на эту кнопку, покажи ему всплывающее окно».

**Для чего тебе это знать**

Понимание этой схемы — фундамент. Когда ты будешь просить ИИ написать код, ты будешь четко понимать: этот кусок отвечает за структуру (HTML), этот за красоту (CSS), а этот за логику (JS). Это сильно сократит количество твоих ошибок и сэкономит время на объяснение ИИ, чего ты хочешь.

# 1.2. Базовый HTML: семантика и структура страницы

Если HTML — это скелет сайта, то тебе нужно понимать, из каких «костей» он состоит. Не пугайся, учить сотни тегов не придется. В вайбкодинге тебе достаточно знать логику, а рутину напишет ИИ.

**Что такое теги**

HTML состоит из тегов. Тег — это команда в угловых скобках. Большинство тегов работают в паре: открывающий и закрывающий. Закрывающий отличается наличием слэша.

Например, чтобы создать заголовок, мы используем тег h1. Выглядит это так:

```
<h1>Привет, мир!</h1>
```

Браузер видит это и понимает: «Ага, тут нужно показать крупный жирный текст». Всё, что находится между открывающим и закрывающим тегом, — это содержимое элемента.

**Структура страницы (каркас)**

Любая правильная HTML-страница имеет жесткую структуру. Представь книгу: у нее есть обложка, оглавление и сама история. У сайта так же.

Вот минимальный каркас, который ИИ будет генерировать постоянно:

```
<!DOCTYPE html>
<html>
<head>
    <title>Мой сайт</title>
</head>
<body>
    <h1>Главный заголовок</h1>
    <p>Текст параграфа.</p>
</body>
</html>
```

Разберем по полочкам:

- `<!DOCTYPE html>` : говорит браузеру, что перед ним современный HTML.
- `<head>` : это служебная зона. Здесь лежит информация для браузера и поисковиков. Пользователь ее не видит. Например, тег `title` задает название вкладки.
- `<body>` : это тело страницы. Всё, что ты видишь на экране (тексты, картинки, кнопки), лежит здесь.

**Что такое семантика (и почему это важно)**

Раньше программисты верстали сайты с помощью тега div (от английского division — блок). Это универсальная коробка. Всё на свете можно было запихнуть в div.

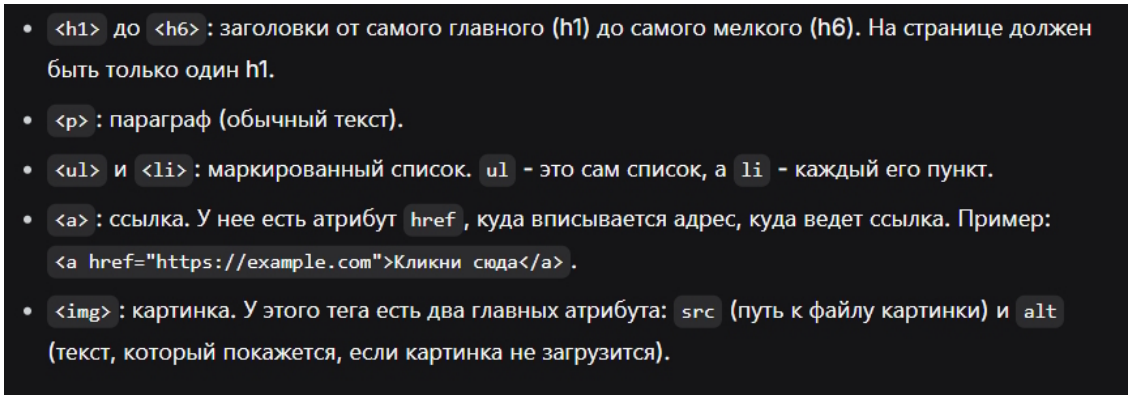
Но потом придумали семантику. Семантические теги — это теги, которые называют вещи своими именами. Они не меняют внешний вид (визуально это тот же блок), но они объясняют браузеру, поисковикам (и ИИ!) смысл этого блока.

- Тег `<header>` - шапка сайта.
- Тег `<footer>` - подвал (низ) сайта.
- Тег `<nav>` - навигация (меню).
- Тег `<main>` - основное содержание.
- Тег `<article>` - статья или карточка товара.

Для чего это знать? Когда ты будешь писать промпт для ИИ, ты можешь сказать: «Сделай мне семантическую верстку». ИИ поймет, что нужно использовать правильные теги, а не просто кучу div. Это сделает код чище, а сайт — дружелюбнее к поисковикам.

**Как выглядит текст на странице**

Для текста есть свои теги. Самые частые:



- `<h1>` до `<h6>`: заголовки от самого главного (h1) до самого мелкого (h6). На странице должен быть только один h1.
- `<p>`: параграф (обычный текст).
- `<ul>` и `<li>`: маркированный список. `ul` - это сам список, а `li` - каждый его пункт.
- `<a>`: ссылка. У нее есть атрибут `href`, куда вписывается адрес, куда ведет ссылка. Пример: `<a href="https://example.com">Кликни сюда</a>`.
- `<img>`: картинка. У этого тега есть два главных атрибута: `src` (путь к файлу картинки) и `alt` (текст, который покажется, если картинка не загрузится).

**Суть для вайбкодера**

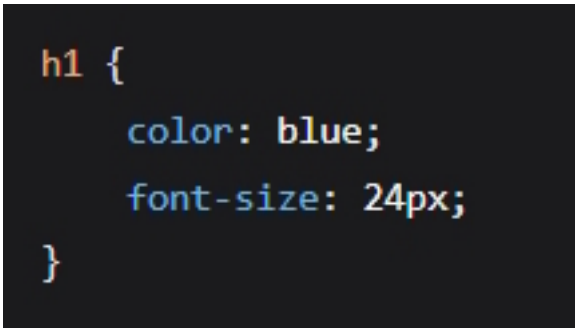
Тебе не нужно помнить точный синтаксис каждого тега. Тебе нужно понимать саму логику: у сайта есть структура, и она состоит из вложенных друг в друга блоков. Когда ты будешь просить ИИ сделать сайт, просто описывай структуру словами: «Мне нужна шапка с меню, посередине статья с картинкой и заголовком, а внизу подвал с контактами». ИИ сам подберет нужные HTML-теги.

# 1.3. Базовый CSS: Flexbox, Grid и адаптивность

Если HTML — это скелет, то CSS — это одежда и макияж. Это язык стилей, который отвечает за то, как элементы выглядят и где они располагаются на экране. В CSS мы пишем правила: «Найди такой-то элемент и сделай с ним вот это».

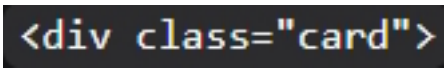
**Как устроен CSS**

Правило CSS состоит из селектора (кого выбираем) и блока со свойствами (что делаем). Выглядит это так:

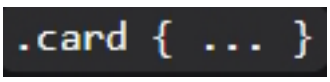



В данном случае мы говорим браузеру: «Найди все заголовки h1 на странице, сделай их текст синим (blue), а размер шрифта установи 24 пикселя (24px)».

Селекторы могут быть разными. Можно выбрать все абзацы (p), а можно дать конкретному элементу класс (например,




и стилизовать только его. В CSS класс обозначается точкой:




**Две главные системы раскладки: Flexbox и Grid**

Раньше в CSS было очень сложно расположить элементы рядом друг с другом (например, сделать карточки товаров в ряд). Приходилось использовать костыли. Но сейчас есть две мощные технологии, которые решают все проблемы с расположением.

**Flexbox (Флексбокс)**

Это одномерная система. Она идеально подходит для выравнивания элементов в одну линию (в строку или в колонку) и управления свободным пространством.

Представь, что у тебя есть меню навигации, и тебе нужно, чтобы логотип был слева, а кнопки — справа. Flexbox справляется с этим в два счета.

Чтобы включить флексбокс, контейнеру (родителю) задают свойство:

```
.container {
    display: flex;
    justify-content: space-between;
}
```

Свойство justify-content управляет выравниванием по горизонтали. Значение space-between раздвигает элементы по краям контейнера.

**Grid (Грид)**

Это двумерная система. Она нужна для создания сложных сеток (одновременно по горизонтали и вертикали). Идеально для галерей, портфолио или каталогов. Ты буквально рисуешь таблицу в CSS и говоришь: «Здесь будет три колонки и две строки».

**Адаптивность (как сделать сайт удобным на телефоне)**

Пользователи заходят на сайты с огромных мониторов, ноутбуков, планшетов и маленьких экранов смартфонов. Адаптивность — это способность сайта подстраиваться под размер экрана.

В CSS это делается с помощью **медиазапросов**. Мы говорим браузеру: «Если экран пользователя меньше определенной ширины, примени вот эти стили».

```
@media (max-width: 600px) {
    .container {
        flex-direction: column;
    }
}
```

Этот код означает: «Если ширина экрана меньше 600 пикселей (то есть это, скорее всего, телефон), то перестрой элементы из строки в колонку, чтобы они не сжимались в нечитаемую кашу».

**Что нужно знать вайбкодеру**

Тебе не нужно заучивать все свойства CSS. Их тысячи. Твоя задача — понимать, что CSS отвечает за внешний вид, и знать о существовании Flexbox и Grid.

Когда ты попросишь ИИ: «Сделай мне красивую адаптивную галерею с картинками в три колонки, чтобы на телефоне она перестраивалась в одну», — ИИ сам напишет нужный код с использованием Grid и медиазапросов.

# 1.4. Основы JavaScript: переменные, функции, массивы и объекты

Если HTML — это скелет, а CSS — одежда, то JavaScript (сокращенно JS) — это мозг и мышцы. Это язык программирования, который оживляет сайт. Он умеет реагировать на клики, считать, запоминать информацию и менять то, что видит пользователь.

Для гуманитария это звучит страшно, но на самом деле в основе JS лежат всего четыре простые концепции. Понимать их нужно для того, чтобы говорить с ИИ на одном языке.

**Переменные (коробки для данных)**

Переменная — это просто коробка с наклейкой, в которую мы кладем данные. У коробки есть имя, чтобы мы могли потом к ней обратиться.

Например, мы хотим сохранить имя пользователя. Мы создаем переменную name и кладем туда слово «Иван».

```
let name = "Иван";
```

Слово let (или const, если данные не будут меняться) — это команда создать коробку. Внутри коробки name теперь лежит текст «Иван». Если мы потом выведем эту переменную на экран, сайт покажет имя.

**Функции (инструкции или рецепты)**

Функция — это блок кода, который выполняет конкретную задачу. Это как рецепт в кулинарной книге: ты даешь функции продукты (данные), она совершает с ними действия и выдает результат.

Например, функция, которая складывает два числа:

```
function add(a, b) {
    return a + b;
}
```

Здесь мы создали функцию add (сложить). В скобках указаны ингредиенты (параметры a и b). Команда return означает «выдай результат». Если мы вызовем add(2, 3), функция вернет нам 5.

**Массивы (списки)**

Массив — это упорядоченный список чего угодно. Представь список покупок или список дел. В коде он выглядит как список значений в квадратных скобках.

```
let shoppingList = ["хлеб", "молоко", "яйца"];
```

В массиве shoppingList лежат три элемента. Мы можем достать первый из них, обратившись к нему по номеру (индексу). Важно помнить: нумерация в программировании начинается с нуля. Поэтому shoppingList[0] — это «хлеб», а shoppingList[1] — «молоко».

**Объекты (карточки с характеристиками)**

Если массив — это просто список, то объект — это структура, где у каждого значения есть свое имя (ключ). Это как анкета или карточка товара.

```
let user = {
    name: "Иван",
    age: 25,
    isAdmin: true
};
```

В объекте user мы описали человека. У него есть имя (строка), возраст (число) и признак администратора (логическое значение true/false). Мы можем обратиться к возрасту через user.age.

**Что нужно знать вайбкодеру**

Вам не нужно учиться писать сложные алгоритмы на JavaScript. Ваша задача — понимать логику:

Переменные нужны, чтобы что-то запомнить.

Функции нужны, чтобы что-то сделать.

Массивы и объекты нужны, чтобы хранить списки и сложные данные.

Когда вы будете просить ИИ написать код, вы можете объяснять ему задачу обычными словами: «Создай переменную для счета игрока. Сделай функцию, которая будет прибавлять к счету одно очко при клике. И запиши имена всех игроков в массив». ИИ сам переведет это на правильный JS.

# 1.5. Что такое DOM и как скрипты меняют интерфейс

В теме 1.2 мы выяснили, что HTML — это скелет страницы. Когда браузер загружает HTML, он строит из него древовидную структуру. Эта структура называется **DOM** (Document Object Model — объектная модель документа).

Если по-простому: DOM — это то, как браузер видит и запоминает страницу. И самое главное: JavaScript может менять эту структуру на лету, не перезагружая страницу.

**Как устроен DOM**

Представь DOM как семейное дерево или иерархию. Самый главный элемент — это документ (document). Внутри него лежит html, внутри html лежат head и body. Внутри body лежат заголовки, параграфы, картинки и кнопки. Каждый элемент в этом дереве называется **узлом** (node).

Каждый узел — это не просто текст на экране. Для JavaScript это объект, у которого есть свойства и методы (возможности). Например, узел «знает», какой у него текст, какой цвет, какие у него родители и дети, и какие действия с ним можно сделать.

**Как JS находит элементы**

Чтобы что-то изменить на странице, JavaScript сначала должен найти нужный элемент. Для этого в браузере есть встроенные команды поиска. Самый частый способ — искать по id или по классу.

Например, если в HTML у нас есть кнопка с идентификатором myButton:

```
<button id="myButton">Купить</button>
```

То в JavaScript мы можем найти её с помощью команды document.getElementById:

```
let button = document.getElementById("myButton");
```

Теперь переменная button содержит ссылку на эту кнопку в DOM. С этого момента мы можем ею управлять.

**Как менять интерфейс**

Когда элемент найден, скрипт может делать с ним разные вещи. Вот три самые частые манипуляции:

**Изменить текст.** Свойство textContent позволяет прочитать или переписать текст внутри элемента.

```
button.textContent = "Товар в корзине";
```

**Изменить стиль.** Свойство style дает доступ к CSS-стилям прямо из кода. Можно поменять цвет, размер, скрыть элемент или показать его.

```
button.style.backgroundColor = "green";
```

**Слушать события.** Мы можем сказать браузеру: «Слушай, когда пользователь кликнет на эту кнопку, выполни вот этот код». Для этого используется метод addEventListener.

```
button.addEventListener("click", function() {
    alert("Вы купили товар!");
});
```

В этом примере мы привязали к кнопке событие клика. Как только пользователь нажмет на неё, сработает функция, которая покажет всплывающее окно.

**Суть для вайбкодера**

DOM — это мост между твоим кодом и тем, что видит пользователь. Весь современный интернет построен на том, что скрипты незаметно для нас меняют DOM. Когда ты ставишь лайк в соцсетях, добавляешь товар в корзину или листаешь ленту без перезагрузки страницы — всё это JavaScript меняет DOM в реальном времени.

Тебе не нужно заучивать команды поиска элементов. Твоя задача — понимать этот принцип. Когда ты будешь говорить ИИ: «Сделай так, чтобы при клике на картинку она увеличивалась», — именно ИИ будет искать нужный узел в DOM и менять его свойства через JavaScript.

# 1.6. Терминал и файловая система: минимум для работы

До этого мы говорили о том, что происходит внутри браузера. Но чтобы начать создавать свои проекты, нам нужно научиться работать с файлами на компьютере. И здесь нам понадобится терминал.

**Что такое файловая система**

Твой компьютер (или телефон) — это огромный шкаф с папками. В одних папках лежат документы, в других — картинки, в третьих — программы. Это и есть файловая система.

В Windows ты привык открывать «Проводник», в Mac — «Finder», и кликать по папкам мышкой. Это графический интерфейс. Но у программистов есть еще один способ управлять файлами — текстовый.

**Что такое терминал (или консоль)**

Терминал — это программа, которая позволяет общаться с компьютером с помощью текстовых команд. Вместо того чтобы кликать мышкой, ты пишешь команду и нажимаешь Enter.

Звучит как хакерство из фильмов, но на самом деле это очень удобно и быстро. В вайбкодинге терминал — твой главный инструмент для запуска проектов и установки нужных программ.

В Windows терминал называется **PowerShell** или **Командная строка** (cmd). В Mac — **Terminal** (Терминал). В редакторе VS Code (о котором мы поговорим в следующем модуле) терминал встроен прямо в окно.

**Базовые команды (всего три)**

Тебе не нужно учить сотни команд. Для старта достаточно понимать три вещи:

**Где я нахожусь?**

В терминале ты всегда находишься в какой-то конкретной папке. Чтобы узнать, в какой именно, введи команду: pwd

(В Windows это сокращение от Print Working Directory — печать рабочей директории). Терминал покажет полный путь к папке, в которой ты сейчас находишься.

**Что здесь лежит?**

Чтобы посмотреть список файлов и папок в текущей директории, введи: ls

(В Windows эта команда называется dir). Терминал выведет список всего, что находится в этой папке.

**Как перейти в другую папку?**

Если ты хочешь зайти внутрь папки, используй команду cd (Change Directory — сменить директорию): cd имя_папки

Например, cd Documents переведет тебя в папку с документами. А команда cd .. (две точки) поднимет тебя на уровень выше, обратно в родительскую папку.

**Зачем это нужно вайбкодеру**

Кажется, что это лишнее, ведь есть мышь. Но когда ты начнешь работать с ИИ-ассистентами и реальными проектами, без терминала не обойтись.

Чтобы запустить локальный сервер, скачать библиотеку, сохранить код в облако (GitHub) или собрать приложение для телефона — ты будешь вводить команды в терминале. ИИ будет давать тебе готовые команды, которые нужно просто скопировать и вставить. Твоя задача — понимать, что ты не просто так вводишь текст, а отдаешь команды своему компьютеру, и видеть, в какой папке ты находишься.

# 2.1. Среда IDE (VS Code) и ИИ-ассистенты: куда вставлять код

В первом модуле мы разобрались с теорией: как устроен интернет, из чего состоит сайт и где хранятся файлы. Теперь пришло время инструментов. Чтобы создавать проекты, тебе понадобится специальная программа — редактор кода.

**Что такое VS Code и зачем он нужен**

Ты можешь создавать сайты в обычном Блокноте. Но это как писать книгу на салфетках: технически возможно, но неудобно. Редактор кода (IDE, Integrated Development Environment) — это Word для программиста.

Самый популярный и бесплатный редактор в мире — **Visual Studio Code** (или просто VS Code). Он работает на Windows, Mac и Linux.

Почему его любят:

Он подсвечивает разными цветами HTML, CSS и JavaScript, чтобы код было легче читать.

Он подсказывает, если ты допустил опечатку.

В него встроен терминал (о котором мы говорили в теме 1.6).

К нему можно подключить искусственный интеллект прямо в окно редактора.

**Куда вставлять код (структура проекта)**

Когда ты создаешь проект, ты не сваливаешь все файлы в одну кучу. В VS Code ты создаешь папку проекта, а внутри нее — файлы.

Главный файл, с которого всё начинается, называется index.html. Это точка входа.

Если ты попросишь ИИ сгенерировать простую страницу, он выдаст тебе код. Твоя задача:

Создать в VS Code файл index.html.

Скопировать код от ИИ и вставить его в этот файл.

Сохранить (Ctrl+S) и открыть этот файл в браузере, чтобы увидеть результат.

По мере усложнения проекта файлов станет больше: отдельный файл для стилей (style.css), отдельный для логики (script.js), плюс папка для картинок. Но суть та же: ИИ пишет код, ты распределяешь его по файлам.

**ИИ-ассистенты: твои цифровые соавторы**

Вайбкодинг невозможен без искусственного интеллекта. Сегодня существует два основных способа работы с ним.

**Способ 1: Чат (ChatGPT, Claude, DeepSeek)**

Это когда ты открываешь сайт ИИ в одном окне, а VS Code — в другом. Ты описываешь задачу, ИИ выдает код в чате, ты его копируешь и вставляешь в редактор.

Плюс: это бесплатно и универсально. Можно задавать любые вопросы.

Минус: нужно постоянно переключаться между окнами и копировать код туда-сюда.

**Способ 2: Встроенные ассистенты (Cursor, GitHub Copilot, расширения для VS Code)**

Это плагины или отдельные редакторы (как Cursor), которые встраивают ИИ прямо в код. ИИ видит весь твой проект, понимает контекст и может писать код прямо в файле, пока ты печатаешь.

Плюс: это невероятно быстро. Ты можешь выделить кусок кода и сказать: «Перепиши это, чтобы работало быстрее», и он сделает это на месте.

Минус: такие инструменты чаще всего платные (по подписке).

**Что выбрать новичку**

Если ты только начинаешь, начни со **Способа 1**. Скачай VS Code, открой ChatGPT или DeepSeek, и работай через копирование и вставку. Это позволит тебе понять сам процесс, не тратя деньги на подписки.

Когда почувствуешь, что тебе надоело переключать окна, и захочешь, чтобы ИИ сам вставлял код в файлы, переходи на Cursor или устанавливай Copilot.

**Суть для вайбкодера**

VS Code — это твой рабочий стол. ИИ — это твой переводчик и программист. Ты выступаешь в роли архитектора: придумываешь идею, описываешь её ИИ, получаешь код, вставляешь его в правильные файлы и проверяешь результат в браузере. Не бойся экспериментировать: в VS Code невозможно сломать компьютер, ты всегда можешь удалить файл и начать заново.

# 2.2. Введение в Git и GitHub: контроль версий с первого дня

Представь, что ты пишешь важный документ в Word. Ты сохраняешь его как «Документ_финал», потом «Документ_финал_2», потом «Документ_финал_настоящий_исправленный». Знакомо? В программировании так делать нельзя: в проекте десятки файлов, и запутаться в них очень легко. Для этого придумали систему контроля версий.

**Что такое Git**

**Git** — это программа, которая работает на твоем компьютере и запоминает историю изменений. Она делает «снимки» твоего проекта (они называются коммитами). Если ты что-то сломал, ты всегда можешь откатиться к предыдущему снимку, где всё работало.

Представь, что Git — это машина времени для твоего кода. Ты работаешь, сохраняешь промежуточные результаты, и в любой момент можешь вернуться на шаг назад.

**Что такое GitHub**

**GitHub** — это облачный сайт, где хранятся эти самые проекты. Это социальная сеть для программистов, но главная его функция — резервное копирование. Если твой ноутбук сгорит, ты не потеряешь код, потому что он сохранен на GitHub.

Связка работает так: ты пишешь код на компьютере, Git запоминает изменения, а потом ты отправляешь эти изменения на GitHub (в облако).

**Зачем это гуманитарию**

В вайбкодинге ИИ будет генерировать много кода. ИИ может ошибиться или сломать то, что уже работало. Без Git ты не поймешь, на каком этапе всё сломалось, и будешь вручную удалять куски кода. С Git ты просто скажешь: «Верни всё как было час назад», и проект восстановится.

Кроме того, GitHub — это твое портфолио. Когда ты захочешь показать свой проект, ты просто скинешь ссылку на GitHub.

**Как это работает в VS Code (минимум действий)**

Тебе не нужно учить сложные команды в терминале. В VS Code есть встроенная вкладка для работы с Git. Она выглядит как иконка с тремя кружочками, соединенными линиями (на левой панели).

Обычно процесс выглядит так:

Ты создал проект и хочешь его сохранить.

Открываешь вкладку Git в VS Code. Ты видишь список измененных файлов.

Пишешь короткое сообщение (например, «Сделал шапку сайта»).

Нажимаешь галочку (Commit). Снимок сохранен!

Если ты привязал GitHub, нажимаешь кнопку «Sync Changes» (или «Publish Branch»), и код улетает в облако.

**Суть для вайбкодера**

Не пугайся слов «контроль версий» и «репозиторий». Воспринимай Git и GitHub как кнопку «Сохранить» с возможностью отмены. Твоя задача — после каждого удачного шага (сделали кнопку, сверстали страницу) нажимать «Сохранить снимок» в VS Code. Если ИИ на следующем шаге всё сломает, ты сможешь безопасно вернуться к последнему рабочему состоянию.

# 2.3. Как формулировать техническое задание для ИИ (TDD-подход)

Вайбкодинг часто понимают неправильно. Новички думают, что это магия: написал ИИ «Сделай мне крутой сайт» — и всё заработало. На самом деле всё наоборот. ИИ — это очень талантливый, но абсолютно глухой исполнитель. Если ты дашь ему размытое задание, он сделает что-то не то.

Чтобы получить хороший результат, нужно использовать подход, который программисты называют TDD (Test-Driven Development — разработка через тестирование). Звучит сложно, но на практике это означает всего одно правило: **сначала описываем, что должно получиться, и только потом просим ИИ написать код**.

**Почему «Сделай красиво» не работает**

Представь, что ты нанял строителя и сказал ему: «Построй мне красивый дом». Скорее всего, ты получишь что-то странное, потому что у каждого свое понятие о красоте.

То же самое с ИИ. Если ты напишешь: «Сделай мне страницу с товарами», ИИ начнет угадывать. Он сделает список в одну колонку, или добавит лишние поля, или выберет ядовитые цвета. Тебе придется переделывать это десять раз.

**Как выглядит правильное ТЗ (промпт)**

Хорошее техническое задание для ИИ состоит из четырех блоков.

**1. Роль**

Скажи ИИ, кем он должен быть. Это настраивает его на нужный лад.

Пример: «Ты — опытный веб-разработчик и эксперт по пользовательским интерфейсам».

**2. Контекст и цель**

Объясни, что мы делаем и для кого.

Пример: «Я делаю сайт-визитку для фотографа. Целевая аудитория — молодожены, которые ищут фотографа на свадьбу. Сайт должен вызывать доверие и выглядеть современно».

**3. Конкретные требования (структура)**

Опиши, из чего состоит страница, шаг за шагом.

Пример:
Вверху шапка с логотипом и кнопкой «Связаться».
Ниже большой баннер с фотографией и заголовком.
Дальше галерея из 6 фотографий в сетке (3 колонки).
Внизу подвал с контактами и ссылками на соцсети.

**4. Технические ограничения (если есть)**

Укажи, какие технологии использовать и чего делать нельзя.

Пример: «Используй только чистый HTML и CSS, без фреймворков. Используй Flexbox или Grid для сетки. Сделай сайт адаптивным, чтобы на телефоне галерея перестраивалась в одну колонку».

**Главное правило: один шаг за раз**

Самая частая ошибка новичков — пытаться решить всё одной гигантской задачей. Если ты попросишь ИИ «Сделай мне интернет-магазин с корзиной, оплатой и личным кабинетом», он выдаст тебе сотни строк кода, в которых ты запутаешься. И скорее всего, там будут ошибки.

Работай итерациями (маленькими шагами).

Сначала попроси сделать только шапку сайта.

Убедись, что она выглядит правильно.

Потом попроси добавить блок с товарами.

Потом — кнопку «Купить».

Каждый маленький шаг легко проверить и легко откатить, если что-то пошло не так.

**Суть для вайбкодера**

ИИ не читает мысли. Чем точнее и подробнее ты опишешь структуру (какие блоки, в каком порядке, какого цвета, как они должны себя вести при нажатии), тем меньше раз ты будешь переделывать работу. Если результат тебя не устроил, не начинай всё заново. Просто напиши ИИ: «Ты сделал галерею в одну колонку, а мне нужно три колонки на компьютере и одна на телефоне. Переделай только этот блок».

# 2.4. Итеративный подход: дробление задач на мелкие шаги

В предыдущей теме мы выяснили, что ИИ нужно давать четкое техническое задание (ТЗ). Но даже самое подробное ТЗ не сработает, если ты попытаешься сделать всё сразу. В этой теме мы разберем главный принцип успешной работы с ИИ — итеративный подход.

**Почему большие задачи убивают проект**

Представь, что ты просишь ИИ: «Напиши мне полноценный сайт для онлайн-школы с расписанием, личным кабинетом и оплатой».

Что произойдет? ИИ сгенерирует огромный массив кода (возможно, сотни строк). Скорее всего, там будут ошибки, несовместимые детали или код, который ты просто не поймешь. Ты скопируешь его, вставишь, откроешь браузер и увидишь белый экран. И ты не поймешь, где именно ошибка: в верстке, в логике или в подключении файлов.

В вайбкодинге работает правило: **чем больше задача, тем выше шанс провала**.

**Что такое итерация (маленький шаг)**

Итерация — это один законченный цикл работы. Ты делаешь один маленький шаг, проверяешь его, и только потом переходишь к следующему.

Вместо гигантского сайта онлайн-школы, ты разбиваешь работу на итерации:

**Итерация 1:** Сделать пустую страницу с шапкой и заголовком «Добро пожаловать».

**Итерация 2:** Добавить под заголовком блок с расписанием занятий (просто статичный список).

**Итерация 3:** Добавить форму входа (логин и пароль).

**Итерация 4:** Сделать так, чтобы при нажатии на кнопку «Войти» появлялось всплывающее окно.

Каждый шаг маленький. Если после первой итерации шапка съехала вбок, ты сразу это видишь и просишь ИИ: «Шапка съехала, исправь». Ты не тратишь время на поиск ошибки в сотнях строк кода.

# 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.