### \*\*JavaScript под капотом: как всё устроено\*\*

JavaScript — это язык, который интерпретируется (исполняется) браузером или сервером (например, Node.js). Когда ты открываешь сайт, браузер получает JS-код, проходит по нему строка за строкой, выполняет этот код и обеспечивает работу интерактивных элементов.

---

### \*\*Как это происходит подробно\*\*

#### \*\*1. Движок JavaScript\*\*

- Основная часть браузера, которая отвечает за выполнение JS, называется \*\*движком\*\*.

- Chrome: V8

- Firefox: SpiderMonkey

- Safari: JavaScriptCore

- Edge: Chakra (старый)

- На сервере (Node.js) — тоже V8.

##### \*\*Процесс:\*\*

1. \*\*Парсинг (разбор):\*\*

JS-код преобразуется в структуру данных — \*абстрактное синтаксическое дерево\* (AST).

2. \*\*Компиляция:\*\*

AST компилируется в “байткод” (более быстрый для исполнения формат).

3. \*\*Исполнение:\*\*

Байткод исполняется в специальной виртуальной машине, а самые часто используемые куски кода \*дополнительно компилируются\* в машинный код процессора (JIT-компиляция — just-in-time).

---

#### \*\*2. Среда выполнения (Runtime)\*\*

- Движок JS сам по себе умеет только считать, сравнивать и работать с переменными.

- Всё, что связано с вебом (DOM, запросы, таймеры) — это \*веб-API\* браузера.

- Есть \*call stack\* (стэк вызова функций) и \*event loop\* (цикл событий).

---

#### \*\*3. Однопоточность и Event Loop\*\*

\*\*JavaScript — однопоточный язык.\*\*

Это значит, что он делает только одну вещь в один момент времени.

##### \*\*Асинхронность реализована так:\*\*

- Пока основной поток кода выполняется, медленные операции (запросы, таймеры, события пользователя) уходят во \*внешние API браузера\*.

- Когда они закончатся — результаты возвращаются обратно через цикл событий (\*\*event loop\*\*).

- \*Callback queue\* — очередь, где лежат все задания, ждущие своего исполнения.

\*\*Пример:\*\*

```js

console.log("Привет");

setTimeout(() => console.log("Мир"), 0);

console.log("JS");

```

\*\*Результат:\*\*

```

Привет

JS

Мир

```

---

#### \*\*4. Память и сборка мусора\*\*

- Записывая значения в переменные, объекты, массивы — мы используем оперативную память.

- Сборщик мусора (`Garbage Collector`) автоматически удаляет неиспользуемые объекты, чтобы память очищалась.

---

#### \*\*5. Взаимодействие с браузером\*\*

- Через специальные объекты и методы (например, `document`, `window`, `alert`), которые браузер предоставляет движку.

- JS вызывает методы браузера (например, изменить HTML через DOM) — браузер обновляет отображение.

---

#### \*\*6. Модули, импорт и экспорт\*\*

- Современные движки поддерживают раздельную загрузку кода — можно разбивать приложение на "модули" и динамически их подгружать.

---

## \*\*Техническая схема (упрощённо)\*\*

```mermaid

graph LR

A[JavaScript код] --> B[Парсер: AST]

B --> C[Компилятор: Байткод]

C --> D[JIT-компилятор: Машинный код]

D --> E[Выполнение кода]

E --> F[Web API, Event loop, Callback queue]

F --> E

```

---

## \*\*Таблица: движки и платформы\*\*

| Платформа | Движок | Особенности |

|--------------|---------------|-----------------------------------------|

| Chrome/Node | V8 | Cамый быстрый, много оптимизаций |

| Firefox | SpiderMonkey | Первый движок JS |

| Safari | JavaScriptCore| Быстрый, родной для Apple |

| IE/старый Edge| Chakra | Уже не развивается |

---

## \*\*Краткое резюме\*\*

- \*\*Движок JS\*\*: Парсит, компилирует и исполняет код.

- \*\*Web API браузера\*\*: Позволяют работать с DOM, сетями, таймерами и т.п.

- \*\*Event loop\*\*: Организует асинхронные действия и работу с очередями.

- \*\*Сборка мусора\*\*: Очищает неиспользуемую память.

- \*\*Вся магия интерактива\*\* — результат взаимодействия движка, API, event loop.

\*Если хочется увидеть «живую» демонстрацию работы цикла событий, js движка или памяти — дай знать, подскажу интерактивные примеры!\*