**Асинхронность, таймеры, fetch, промисы**

**9.1. Что такое асинхронность?**

JavaScript — **однопоточный язык**, но с возможностью выполнять **асинхронные операции**:

* Таймеры (setTimeout)
* Запросы на сервер (fetch)
* Ожидание загрузки (onload)
* События пользователя (click, keydown)

Асинхронность = **код продолжает выполняться, не дожидаясь ответа**.

**9.2. Таймеры: setTimeout и setInterval**

**setTimeout — задержка**

js

КопироватьРедактировать

setTimeout(() => {

console.log("Привет через 2 секунды");

}, 2000);

**setInterval — повторение**

js

КопироватьРедактировать

setInterval(() => {

console.log("Каждые 3 секунды");

}, 3000);

Остановка:

js

КопироватьРедактировать

let timer = setInterval(...);

clearInterval(timer);

**9.3. Колбэки и их ловушка (callback hell)**

js

КопироватьРедактировать

function step1(cb) {

console.log("Шаг 1");

cb();

}

step1(() => {

console.log("Шаг 2");

});

⚠️ При вложенных вызовах → **хаос**:

js

КопироватьРедактировать

step1(() => {

step2(() => {

step3(() => {

// ужас

});

});

});

**9.4. Промисы (Promise)**

Объект, который представляет **результат асинхронной операции в будущем**.

js

КопироватьРедактировать

let promise = new Promise((resolve, reject) => {

setTimeout(() => {

resolve("Готово!");

}, 2000);

});

**9.5. Обработка промисов: then, catch, finally**

js

КопироватьРедактировать

promise

.then(result => console.log(result)) // "Готово!"

.catch(error => console.log("Ошибка", error))

.finally(() => console.log("Всегда"));

**9.6. Цепочка промисов**

js

КопироватьРедактировать

getData()

.then(data => transform(data))

.then(result => display(result))

.catch(err => console.error(err));

**9.7. fetch() — запрос на сервер**

js

КопироватьРедактировать

fetch("https://jsonplaceholder.typicode.com/posts/1")

.then(response => response.json())

.then(data => console.log(data));

**9.8. Асинхронные функции: async/await**

js

КопироватьРедактировать

async function loadData() {

let res = await fetch("https://api");

let data = await res.json();

console.log(data);

}

👍 Читабельно, как обычный код  
⚠️ Можно использовать только внутри async-функций

**9.9. Ошибки в async/await: try...catch**

js

КопироватьРедактировать

async function getData() {

try {

let res = await fetch("https://api");

let json = await res.json();

console.log(json);

} catch (err) {

console.log("Ошибка!", err);

}

}

**9.10. Глоссарий**

| **Термин** | **Значение** |
| --- | --- |
| Асинхронность | Код, который выполняется позже, не блокируя поток |
| Таймер | Функция задержки (setTimeout, setInterval) |
| Callback | Функция, переданная в другую функцию |
| Promise | Обещание результата в будущем |
| fetch | Запрос к внешнему серверу |
| async/await | Современный способ работы с асинхронностью |

**9.11. Частые ошибки**

❌ Забыли await → промис, а не данные  
❌ Нет return внутри .then() → цепочка ломается  
❌ Обработка ошибки вне try/catch  
❌ Используют await вне async

**9.12. Вопросы для самопроверки**

1. В чём разница между setTimeout и setInterval?
2. Что такое промис? Когда он используется?
3. Что делают then, catch, finally?
4. Как работает async/await?
5. Как правильно обрабатывать ошибку в fetch?