Тема 11. Ошибки и отладка.

хекслет колледж

Цель занятия:

Сформировать у студентов понимание видов ошибок в JavaScript,
научить читать сообщения об ошибках и использовать инструменты браузера для поиска и исправления ошибок в коде.

Учебные вопросы:

1. Понятие ошибки в программе. Виды ошибок в JavaScript
2. Сообщения об ошибках в консоли браузера
3. try...catch — обработка ошибок
4. Пользовательские классы ошибок. Оператор throw.
5. Использование console для отладки
5. Инструменты разработчика браузера (DevTools)

1. Понятие ошибки в программе. Виды ошибок в JavaScript

Ошибка в программе — это ситуация, при которой программа:

- не запускается,
- завершает работу некорректно,
- или работает не так, как задумано разработчиком.

Ошибки являются нормальной частью процесса разработки и возникают даже у опытных программистов.
Задача разработчика — научиться находить и исправлять ошибки, а не избегать их полностью.

Основные причины появления ошибок:

- опечатки в коде;
- неправильное использование синтаксиса языка;
- работа с несуществующими переменными или объектами;
- неверная логика программы;
- ошибочные предположения о входных данных.

Виды ошибок в JavaScript:

Синтаксические ошибки (Syntax Error)

Синтаксическая ошибка возникает, если нарушены правила языка JavaScript.
Программа не может быть выполнена до исправления такой ошибки.
Примеры причин:

- пропущена скобка;
- забыта кавычка;
- неверно написано ключевое слово.

Ошибки времени выполнения (Runtime Error)

Ошибка времени выполнения возникает во время работы программы,
когда синтаксис написан корректно, но операция невозможна.
Примеры причин:

- обращение к несуществующей переменной;
- попытка вызвать несуществующую функцию;
- доступ к свойству undefined или null.

Логические ошибки

Логическая ошибка — наиболее сложный вид ошибки.

Программа:

- запускается,
- не выдаёт сообщений об ошибке,
- но работает неправильно.

Вывод:

- Ошибки неизбежны в программировании.
- В JavaScript различают синтаксические, runtime и логические ошибки.
- Не все ошибки выводятся в консоль — логические ошибки нужно искать самостоятельно.
- Умение анализировать ошибки — ключевой навык разработчика.

2. Сообщения об ошибках в консоли браузера

Консоль браузера — основной инструмент разработчика для:

- просмотра сообщений об ошибках;
- вывода отладочной информации;
- анализа работы JavaScript-кода.

Все ошибки JavaScript автоматически выводятся в консоль, что позволяет быстро определить место и причину сбоя.

Как открыть консоль?

В большинстве браузеров:

- клавиша F12
- или сочетание Ctrl + Shift + I
- вкладка Console

Что происходит при ошибке?

Если в коде возникает ошибка:

- выполнение скрипта может быть остановлено;
- в консоль выводится сообщение об ошибке, выделенное красным цветом;
- указывается файл и номер строки, где возникла ошибка.

Структура сообщения об ошибке

Типичное сообщение об ошибке в консоли содержит:

1. Тип ошибки. Например:

- SyntaxError
- ReferenceError
- TypeError

2. Описание ошибки. Краткое пояснение, что именно пошло не так.

3. Источник ошибки:

- имя файла;
- номер строки;
- номер столбца (не всегда).

Примеры распространённых сообщений

ReferenceError.
Возникает при обращении к несуществующей переменной:
console.log(a);
Сообщение:
ReferenceError: a is not defined

TypeError.

Возникает при попытке выполнить недопустимую операцию с типом данных:

```
let num = 10;
num();
```

Сообщение:

TypeError: num is not a function

SyntaxError

Возникает при нарушении синтаксиса:

```
if (true {
    console.log("Ошибка");
}
```

Сообщение:

SyntaxError: Unexpected token '{'

Как читать сообщение об ошибке?

Алгоритм работы с ошибкой:

- Посмотреть тип ошибки.
- Прочитать описание ошибки.
- Перейти по ссылке на строку кода.
- Проверить переменные, скобки, условия.

Вывод:

- Консоль браузера — основной инструмент диагностики ошибок.
- Сообщение об ошибке всегда содержит тип и место возникновения.
- Умение читать сообщения об ошибках значительно ускоряет отладку.
- Ошибка в консоли — подсказка, а не «поломка программы».

3. try...catch — обработка ошибок

Зачем нужна обработка ошибок?
Не все ошибки можно предотвратить заранее.
Иногда ошибка возникает из-за:

- некорректных данных,
- действий пользователя,
- неожиданного состояния программы.

Чтобы программа не прекращала работу аварийно, в JavaScript используется механизм обработки ошибок.

Что такое try...catch?

try...catch — это конструкция, которая позволяет:

- проверить выполнение участка кода,
- перехватить ошибку, если она возникнет,
- выполнить альтернативные действия вместо остановки программы.

Общий синтаксис try...catch:

```
try {
    // код, в котором может возникнуть
ошибка
} catch (error) {
    // код, который выполнится при ошибке
}
```

Как работает try...catch?

-JavaScript выполняет код в блоке **try**.
-Если ошибки нет — блок catch пропускается.
-Если возникает ошибка:

- выполнение блока **try** прекращается;
- управление передаётся в **catch**;
- ошибка сохраняется в переменной **error**.

Пример перехвата ошибки:

```
try {
    console.log(user.name);
} catch (error) {
    console.log("Произошла ошибка");
}
```

Если переменная user не определена:

- ошибка не остановит выполнение программы;
- в консоль будет выведено сообщение из catch.

Объект ошибки

Переменная **error** содержит информацию об ошибке:

- **error.name** — тип ошибки;
- **error.message** — описание ошибки.

Пример:

```
catch (error) {
    console.log(error.name);
    console.log(error.message);
}
```

Использование finally

Блок **finally** выполняется всегда, независимо от наличия ошибки.

```
try {
    console.log("Начало");
} catch (error) {
    console.log("Ошибка");
} finally {
    console.log("Конец выполнения");
}
```

Важно помнить:

- try...catch не ловит синтаксические ошибки.
- Он используется для обработки ошибок времени выполнения.
- Нельзя применять try...catch для скрытия ошибок — он нужен для корректного реагирования.

Вывод:

- **try...catch** предотвращает аварийное завершение программы.
- Позволяет управлять поведением кода при ошибках.
- Используется для повышения устойчивости приложения.
- Часто применяется при работе с данными и пользовательским вводом.

4. Пользовательские классы ошибок. Оператор throw.

В процессе выполнения программы могут возникать ситуации, при которых дальнейшая работа кода невозможна или некорректна. Для таких случаев в JavaScript используется механизм исключений: оператор throw и объекты ошибок, включая пользовательские классы ошибок.

Оператор throw

Оператор **throw** используется для явного создания ошибки в программе.

Назначение **throw:**

- сигнализировать о недопустимой ситуации;
- остановить выполнение текущего кода;
- передать управление в блок catch.

Синтаксис:

throw new Error("Сообщение об ошибке");

или

throw Error("Сообщение об ошибке");

После выполнения **throw**:

- выполнение текущей функции прекращается;
- программа ищет ближайший блок catch;
- если catch отсутствует, ошибка выводится в консоль.

Обработка выброшенной ошибки (try...catch)

```
try {
    divide(10, 0);
} catch (error) {
    console.log("Ошибка:", error.message);
}
```

Если внутри try выполняется throw, выполнение кода сразу переходит в catch.

Пользовательские классы ошибок

Для более сложных проектов можно создавать собственные типы ошибок, расширяя стандартный класс Error.

```
class ValidationError extends Error {
    constructor(message) {
        super(message);
        this.name = "ValidationError";
    }
}
```

Использование пользовательской ошибки

```
function checkAge(age) {
    if (age < 0) {
        throw new ValidationError("Возраст не может быть отрицательным");
    }
}
```

Пользовательские классы ошибок позволяют:

- различать типы ошибок;
- обрабатывать ошибки по-разному;
- делать код более читаемым и поддерживаемым.

Пример:

```
try {
    checkAge(-5);
} catch (error) {
    if (error.name === "ValidationError") {
        console.log("Ошибка ввода:", error.message);
    } else {
        console.log("Другая ошибка");
    }
}
```

Когда что использовать?

- throw Error(...) -Учебные и простые проекты
- Пользовательские классы ошибок - Крупные и сложные проекты

Итоги:

- throw используется для генерации ошибок вручную.
- Лучше выбрасывать объект Error, а не строку или число.
- try...catch позволяет перехватывать и обрабатывать ошибки.
- Пользовательские классы ошибок расширяют возможности обработки ошибок.
- Для начинающих достаточно throw Error(...), но важно понимать принцип.

5. Использование console для отладки

Отладка — процесс поиска и исправления ошибок в программе.
Одним из самых простых и эффективных инструментов отладки является объект console.

Методы **console** позволяют:

- проверять значения переменных;
- отслеживать выполнение кода;
- выявлять логические ошибки.

console.log()

console.log() — основной и самый часто используемый метод.

Используется для вывода:

- значений переменных;
- промежуточных результатов;
- текстовых сообщений.

console.log() применяется:

- при поиске логических ошибок;
- при проверке условий;
- при изучении работы кода шаг за шагом.

Важно:

- после завершения отладки лишние console.log() следует удалять.

Пример:

```
fetch("https://jsonplaceholder.typicode.com/users/1")
    .then(response => response.json())
    .then(data => {
        console.log("Данные, полученные с сервера:", data);
    })
    .catch(error => {
        console.error("Ошибка при загрузке данных:", error);
    });
```

console.warn()

console.warn() выводит предупреждение.
Используется, когда:

- ситуация не является критической ошибкой;
- требуется обратить внимание разработчика.

Сообщения отображаются жёлтым цветом.

Пример. Необязательное поле не пришло с сервера

```
fetch("https://jsonplaceholder.typicode.com/users/1")
    .then(response => response.json())
    .then(data => {
        if (!data.phone) {
            console.warn("Номер телефона отсутствует в данных пользователя");
        }

        console.log("Имя:", data.name);
    });
```

console.error()

console.error() выводит сообщение об ошибке.

Используется:

- для имитации ошибок;
- при обработке исключений;
- для явного указания на проблему.

Сообщения отображаются красным цветом.

Пример:

Ошибка в вычислениях. Результат функции ломает дальнейшую логику (например, вместо числа пришел NaN)

```
if (isNaN(price)) {
  console.error("Ошибка: итоговая цена не является числом (NaN)!", { items });
}
```

Вывод:

- console — простой и эффективный инструмент отладки.
- console.log(), console.warn() и console.error() решают разные задачи.
- Регулярное использование console ускоряет поиск ошибок.
- Отладка — обязательный этап разработки.

6. Инструменты разработчика браузера (DevTools)

Инструменты разработчика браузера (DevTools) — это встроенный набор средств,
который позволяет:

- анализировать HTML и CSS;
- отлаживать JavaScript-код;
- находить и исправлять ошибки;
- проверять производительность и адаптивность сайта.

DevTools доступны во всех современных браузерах (Chrome, Edge, Firefox и др.).

Как открыть DevTools?

Наиболее распространённые способы:

- F12
- Ctrl + Shift + I
- ПКМ на странице → «Просмотреть код» / «Inspect»

Основные вкладки DevTools

1. Elements

Позволяет:

- просматривать структуру HTML-документа;
- видеть стили элементов;
- временно изменять HTML и CSS прямо в браузере.

Важно:
изменения в Elements не сохраняются и используются только для анализа.

2. Console.

Основная вкладка для работы с JavaScript.

Возможности:

просмотр ошибок и предупреждений;

вывод console.log(), console.warn(), console.error();

выполнение JavaScript-кода вручную.

3. Sources

Используется для глубокой отладки JavaScript.

Позволяет:

- просматривать подключённые JS-файлы;
- ставить точки останова (breakpoints);
- выполнять код пошагово;
- отслеживать значения переменных.

4. Network.

Используется для анализа загрузки ресурсов.

Позволяет:

- видеть запросы fetch;
- проверять успешность загрузки данных;
- анализировать время ответа сервера;
- находить ошибки загрузки (404, 500).

5. Application.

Позволяет:

- просматривать localStorage и sessionStorage;
- анализировать cookies;
- проверять данные, сохранённые в браузере.

Вывод:

- DevTools — обязательный инструмент фронтенд-разработчика.
- Console и Sources — ключевые вкладки для JavaScript.
- Умение пользоваться DevTools ускоряет разработку и отладку.
- Практика работы с DevTools — основа профессионального роста.

Итоги лекции:

В JavaScript различают:

- Синтаксические ошибки (SyntaxError) — нарушения правил языка.
- Ошибки времени выполнения (Runtime Error) — некорректные операции во время работы программы.
- Логические ошибки — программа работает, но результат неверный.

Сообщения об ошибках в консоли помогают быстро определить:

- тип ошибки;
- строку и файл, где ошибка возникла;
- причину сбоя.

Конструкция try...catch позволяет:

- перехватывать ошибки времени выполнения;
- предотвращать аварийное завершение программы;
- корректно реагировать на ошибки.

DevTools браузера предоставляет полный набор инструментов для работы с кодом:

- Elements — просмотр и правка HTML/CSS;
- Console — просмотр ошибок, вывод данных, отладка;
- Sources — пошаговое выполнение кода, breakpoints;
- Network — анализ загрузки ресурсов и запросов;
- Application — просмотр localStorage, cookies, данных приложения.

Контрольные вопросы:

- Чем синтаксическая ошибка отличается от ошибки времени выполнения?
- Что такое логическая ошибка и почему её сложно обнаружить?
- Для чего используется консоль браузера?
- Какие данные содержит сообщение об ошибке?
- Зачем используется конструкция try...catch?
- Для чего нужна вкладка Sources?
- Как DevTools помогает находить ошибки в коде?
- Как Network помогает при работе с внешними данными?

Домашнее задание:

https://ru.hexlet.io/courses/js-basics

хекслет колледж

@HEXLY.KZ