

Тема 3. Операторы и выражения



Хекслет колледж

Цель занятия:

Сформировать у студентов понимание операторов JavaScript, их видов и порядка выполнения, а также научить составлять и анализировать выражения, используемые в условиях и вычислениях.

Учебные вопросы:

1. Понятие оператора, операнда и выражения
2. Арифметические операторы
3. Операторы присваивания
4. Операторы сравнения
5. Логические операторы
6. Приоритет операторов
7. Math. Свойства и методы.

1. Понятие оператора, операнда и выражения

Оператор.

Оператор в JavaScript — это специальный символ или ключевое слово, которое выполняет определённое действие над данными.

Операторы указывают:

- что именно нужно сделать с данными,
- какую операцию выполнить.

Примеры операторов:

+ — сложение

- — вычитание

= — присваивание

> — сравнение

&& — логическое И

let sum = 5 + 3;

В данном примере операторы: **=** и **+**.

Операнд

Операнд — это значение, переменная или выражение, над которым оператор выполняет действие.

5 + 3

+ — оператор

5 и **3** — операнды

Виды операторов по количеству операндов

- Унарные (1 операнд):

!isAdmin
typeof age

- Бинарные (2 операнда):

a + b
x === y

- Тернарные (3 операнда, будут изучены позже):

condition ? value1 : value2

Выражение

Выражение — это комбинация операторов и операндов, которая вычисляется и возвращает значение.

Примеры выражений:

10 + 5

age > 18

isLoggedIn && isAdmin

Каждое выражение имеет результат:

- число,
- строку,
- логическое значение.

Связь понятий

Операторы выполняют действия

Операнды — данные, над которыми выполняются действия

Выражения — результат применения операторов к операндам

Пример:

let result = (10 + 5) * 2;

(10 + 5) * 2 — выражение

+ и ***** — операторы

10, 5, 2 — операнды

Итог

Оператор выполняет действие над операндами и формирует выражение, а выражение всегда можно вычислить и получить результат.

2. Арифметические операторы

Арифметические операторы используются для выполнения математических вычислений над числовыми значениями и переменными.

Они применяются:

- при расчётах,
- в формулах,
- при обработке числовых данных.

Основные арифметические операторы

Оператор	Назначение	Пример
+	сложение	$5 + 3$
-	вычитание	$10 - 4$
*	умножение	$6 * 2$
/	деление	$10 / 2$

Оператор остатка от деления %

Оператор % возвращает остаток от деления одного числа на другое.

`10 % 3; // 1`

Используется, например:

- для проверки чётности числа,

Инкремент и декремент

Инкремент (**++**) — увеличивает значение на 1

Декремент (**--**) — уменьшает значение на 1

```
let a = 5;
```

```
a++; // 6
```

```
a--; // 5
```

Префиксная и постфиксная формы

```
let x = 5;  
console.log(++x); // 6
```

```
let y = 5;  
console.log(y++); // 5
```

Префиксная форма (**++x**) — сначала увеличивает, потом возвращает значение.

Постфиксная форма (**x++**) — сначала возвращает значение, потом увеличивает.

Арифметика со строками

Оператор + со строками выполняет конкатенацию:

"10" + 5; // "105"

Остальные арифметические операторы приводят значения к числу:

"10" - 5; // 5

"10" * 2; // 20

Особые числовые значения

NaN — результат некорректного числового
вычисления

Infinity — результат деления на ноль

10 / 0; // Infinity

"abc" / 2; // NaN

Итог:

Арифметические операторы:

- выполняют математические действия,
- работают в выражениях,
- часто используются совместно с переменными и преобразованием типов.
- работают с числами, но оператор **+** может работать и со строками.

3. Операторы присваивания

Оператор присваивания (=) используется для: сохранения значения в переменной, изменения существующего значения переменной. Примеры:

```
let x = 10; // присваиваем 10 переменной x  
let y = x;  // присваиваем значение x переменной y
```

Левое значение (переменная) получает правое значение (число, строку, выражение).

Составные операторы присваивания

Составные операторы позволяют одновременно изменять значение переменной и присваивать его.

Оператор	Описание	Пример
<code>+=</code>	сложение и присваивание	<code>x += 5; // x = x + 5</code>
<code>-=</code>	вычитание и присваивание	<code>x -= 3; // x = x - 3</code>
<code>*=</code>	умножение и присваивание	<code>x *= 2; // x = x * 2</code>
<code>/=</code>	деление и присваивание	<code>x /= 4; // x = x / 4</code>
<code>%=</code>	остаток от деления и присваивание	<code>x %= 3; // x = x % 3</code>

Примеры использования:

```
let score = 10;
```

```
score += 5; // score = 15
```

```
score -= 3; // score = 12
```

```
score *= 2; // score = 24
```

```
score /= 4; // score = 6
```

```
score %= 4; // score = 2
```

Для строк можно использовать +=
для конкатенации:

```
let greeting = "Hello";  
greeting += " World";    // "Hello World"
```

Итог:

- `=` присваивает значение переменной.
- `+=`, `-=`, `*=`, `/=`, `%=` — сокращённые формы присваивания с операцией.
- Составные операторы повышают удобство и читаемость кода.
- Операторы присваивания связывают переменную с выражением и могут одновременно изменять её значение.

4. Операторы сравнения

Операторы сравнения используются для:

- сравнения двух значений,
- получения логического результата: **true** или **false**.
- Они важны для условий, циклов и логики программы.

Основные операторы сравнения

Оператор	Описание	Пример
==	равно (с приведением типов)	5 == "5" → true
===	строго равно (без приведения типов)	5 === "5" → false
!=	не равно (с приведением типов)	5 != "6" → true
!==	строго не равно (без приведения типов)	5 !== "5" → true
>	больше	10 > 5 → true
<	меньше	3 < 7 → true
>=	больше или равно	5 >= 5 → true
<=	меньше или равно	4 <= 5 → true

Разница между `==` и `===`

`==` — выполняет неявное преобразование типов, поэтому сравнение `"5" == 5` вернёт **true**.

`===` — проверяет и значение, и тип, поэтому `"5" === 5` вернёт **false**.

Рекомендуется использовать `===` и `!==`, чтобы избежать неожиданных результатов.

Примеры использования:

```
let a = 10;  
let b = "10";
```

```
console.log(a == b);    // true  
console.log(a === b);  // false  
console.log(a != 5);    // true  
console.log(a > 7);     // true  
console.log(a <= 10);   // true
```

Сравнение строк

Строки сравниваются по Unicode-коду символов.

```
console.log("abc" > "abd");      // false  
console.log("apple" < "banana"); // true
```

Когда JavaScript сравнивает строки, он смотрит на Unicode-код каждого символа.

Символы в таблице Unicode расположены примерно в алфавитном порядке для латиницы, а цифры идут раньше букв.

То есть 'a' < 'b' и 'A' < 'B'.

Сравнение идёт символ за символом слева направо, пока не найдётся первое различие.

Итог:

- Используйте строгие операторы (`===`, `!==`) для надёжности.
- Сравнение всегда возвращает **true** или **false**.
- строки сравниваются посимвольно по кодам символов.
- Для чисел — сравнение как обычно в математике.

5. Логические операторы

Логические операторы позволяют:

- объединять или изменять логические значения (`true / false`),
- строить сложные условия для ветвлений и циклов.

В JavaScript логические операторы всегда возвращают логическое значение.

Основные логические операторы

Оператор	Назначение	Пример
&&	логическое И	true && false → false
	логическое ИЛИ	true false → true
!	логическое НЕ	!true → false

Логическое И (&&)

Возвращает **true**, если оба значения истинны.
Иначе возвращает **false**.

```
let a = true;
```

```
let b = false;
```

```
console.log(a && b); // false
```

```
console.log(true && true); // true
```


Логическое ИЛИ (||)

Возвращает **true**, если хотя бы одно значение истинно.

Только если оба значения **false** — результат false.

```
let a = true;
```

```
let b = false;
```

```
console.log(a || b); // true
```

```
console.log(false || false); // false
```

Логическое НЕ (!)

Меняет логическое значение на противоположное.

```
let isAdmin = true;  
console.log(!isAdmin); // false
```

Особенности логических операторов

В отличие от многих других языков программирования, в JavaScript они не просто возвращают `true` или `false`, а работают по принципу «короткого замыкания» и могут возвращать значения **любых типов**.

в JavaScript любое значение в логическом контексте приводится либо к `true`, либо к `false`.

Falsy (ложные) значения: `false`, `0`, `""` (пустая строка), `null`, `undefined` и `NaN`.

Truthy (истинные) значения: Всё остальное (массивы `[]`, объекты `{}`, любые непустые строки, числа кроме нуля).

Логическое И (&&) — поиск первого «лживого»

Оператор **&&** вычисляет операнды слева направо. Он останавливается на первом «ложном» значении и возвращает его. Если все значения «истинны», он возвращает последнее. Примеры:

```
console.log(0 && "Привет");    // 0 (первое ложное)
console.log("Текст" && null);   // null (первое ложное)
console.log("А" && "Б" && "В"); // "В" (все истины, вернул последнее)
```

Особенности применения:

Его часто используют для коротких проверок (Short-circuit evaluation). Вместо того чтобы писать `if`, можно выполнить код, только если условие истинно:

`isLoggedIn && showDashboard();` — функция выполнится только если пользователь вошел в систему.

Логическое ИЛИ (||) — поиск первого «истинного»

Оператор **||** также работает слева направо.

Он останавливается на первом «истинном» значении и возвращает его.

Если все значения «ложны», он возвращает последнее.

Примеры:

```
console.log(true || "Ошибка"); // true
```

```
console.log("" || "Аноним"); // "Аноним" (пустая строка — false, вернул первое истинное)
```

```
console.log(null || 0 || false); // false (все ложны, вернул последнее)
```

Особенности применения:

Основное применение — задание значений по умолчанию.

const name = userName || "Гость"; — если в `userName` пусто (`null` или пустая строка), в переменную запишется "Гость".

Итог:

- && — логическое И, оба значения должны быть истинными.
- || — логическое ИЛИ, достаточно одного истинного значения.
- ! — логическое НЕ, меняет значение на противоположное.
- Логические операторы позволяют строить условия, которые зависят от нескольких проверок одновременно.

6. Приоритет операторов

Приоритет операторов — это правило, которое определяет порядок выполнения операций в выражении.

Если выражение содержит несколько операторов, JavaScript сначала выполняет операции с более высоким приоритетом, а затем — с более низким. Скобки () позволяют изменить порядок выполнения, повышая приоритет операций внутри них.

Примеры приоритета:

```
let result = 2 + 3 * 4;  
console.log(result); // 14
```

* имеет высший приоритет, чем +
Сначала выполняется $3 * 4 = 12$
Потом $2 + 12 = 14$

```
let result = (2 + 3) * 4;  
console.log(result); // 20
```

Скобки изменили порядок: сначала $2 + 3 = 5$, потом $5 * 4 = 20$

Уровни приоритета операторов

Уровень	Операторы	Пример
Высокий	<code>()</code> , <code>++</code> , <code>--</code>	<code>(2 + 3)</code> , <code>x++</code>
Высокий	Унарные <code>+</code> , <code>-</code> , <code>!</code>	<code>-a</code> , <code>!isAdmin</code>
Средний	<code>*</code> , <code>/</code> , <code>%</code>	<code>5 * 2</code> , <code>10 / 2</code>
Средний-низкий	<code>+</code> , <code>-</code>	<code>2 + 3</code> , <code>5 - 1</code>
Низкий	<code><</code> , <code>></code> , <code><=</code> , <code>>=</code>	<code>a > b</code>
Низкий	<code>==</code> , <code>===</code> , <code>!=</code> , <code>!==</code>	<code>x === y</code>
Очень низкий	<code>&&</code>	<code>true && false</code>
Самый низкий	<code>`</code>	<code>let greeting = `Привет, \${name}!`</code>
Присваивание	<code>=</code> , <code>+=</code> , <code>-=</code> , <code>*=</code> и т.д.	<code>x = 5</code> , <code>x += 2</code>

Чем выше в таблице — тем раньше выполняется операция.
Операции на одном уровне выполняются слева направо.

Практические рекомендации:

- Используйте скобки для ясности и контроля порядка выполнения.
- Помните, что арифметические операции выполняются сначала, потом сравнения, потом логические операции.
- При сложных выражениях лучше разделять их на несколько переменных для читаемости.

Примеры сложных выражений:

```
let a = 5, b = 10, c = 2;  
let result = a + b * c > 20 && b > c;  
console.log(result); // true
```

Разбор по приоритету:

$b * c = 10 * 2 = 20$

$a + (b * c) = 5 + 20 = 25$

$25 > 20 \rightarrow \text{true}$

$b > c = 10 > 2 \rightarrow \text{true}$

$\text{true} \&\& \text{true} \rightarrow \text{true}$

Итог:

- Приоритет операторов определяет, какая операция выполняется раньше.
- Скобки всегда дают возможность явно задать порядок.
- Правильное понимание приоритета предотвращает неожиданные результаты вычислений.

7. Math. Свойства и методы.

Math — это встроенный объект в JavaScript, который предоставляет набор математических функций и констант.

Его методы и свойства можно использовать для выполнения различных математических операций, таких как округление, нахождение максимума/минимума, генерация случайных чисел и многое другое.

Свойства Math

Math.PI — значение числа π (пи),
приблизительно 3.14159.

`console.log(Math.PI); // 3.141592653589793`

Math.E — основание натурального
логарифма (число Эйлера), приблизительно
2.718.

Методы Math

Округление чисел

Math.round(x) — округление числа до ближайшего целого.

```
console.log(Math.round(4.7)); // 5  
console.log(Math.round(4.3)); // 4
```

Math.floor(x) — округление вниз (в меньшую сторону).

Math.ceil(x) — округление вверх (в большую сторону).

Math.trunc(x) — отбрасывает дробную часть числа (не округляет).

Минимум и максимум

Math.min(a, b, ...) — возвращает наименьшее из переданных чисел.

Math.max(a, b, ...) — возвращает наибольшее из переданных чисел.

Случайные числа

Math.random() — возвращает псевдослучайное число от 0 (включительно) до 1 (не включительно).

**console.log(Math.random()); // Например,
0.7324040990734169**

Генерация случайных чисел в диапазоне:

// Случайное число от 0 до 10 (целое)
const randomInt = Math.floor(Math.random() * 10);

Степени и корни

Math.pow(base, exponent) — возведение в степень.

Math.sqrt(x) — квадратный корень.

Math.cbrt(x) — кубический корень.

Абсолютное значение

Math.abs(x) — возвращает абсолютное значение числа (модуль числа).

```
console.log(Math.abs(-10)); // 10
```

Тригонометрические функции

`Math.sin(x)`, `Math.cos(x)`, `Math.tan(x)` — тригонометрические функции (аргумент в радианах).

```
console.log(Math.sin(Math.PI / 2)); // 1  
console.log(Math.cos(0)); // 1
```

Логарифмы

`Math.log(x)` — натуральный логарифм (по основанию e).

`Math.log10(x)` — десятичный логарифм (по основанию 10).

`Math.log2(x)` — логарифм по основанию 2.

Контрольные вопросы:

- Что такое оператор в JavaScript? Приведите пример.
- Что такое операнд? Приведите пример.
- Какие арифметические операторы вы знаете? Для чего используется %?
- В чём разница между `x++` и `++x`?
- Что произойдет при сложении числа и строки с помощью `+`?
- Как работают составные операторы присваивания (`+=`, `-=`, `*=`)? Приведите пример.
- Можно ли использовать `+=` с константой (`const`)? Почему?
- В чём разница между `==` и `===`?
- Что будет результатом сравнения `"10" > 5`? Почему?
- Что делает оператор `!`?
- Что такое приоритет операторов?
- Какие операторы имеют самый низкий приоритет?
- Как скобки `()` влияют на порядок выполнения операций?

Домашнее задание:

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

хекслет колледж

@HEXLY.KZ