Выражения и операторы

Программа на JavaScript

JavaScript программы выполняются браузером. Программа на JavaScript состоит из **набора программных инструкций**, которые должен выполнить компьютер. **Инструкции** JavaScript **выполняются последовательно**, друг за другом, в том порядке, в котором они написаны.

Одна инструкция может располагаться на нескольких строчках.

```
// инструкция if
if (role == "admin") {
     alert('Добро пожаловать!');
}
```

С другой стороны, на одной строке может находиться **несколько инструкций**, если они отделены друг от друга точкой с запятой.

```
var name, login; name = "Max"; login = "master";
```

Инструкции

Инструкция (англ. *statement*) - это программная конструкция, которая выполняет определенные **действия**.

Ниже приведены некоторые **примеры инструкций**:

example_1. Примеры инструкций

```
<script>
    // инструкции создания переменных
    let i;
    const user = { age: 15 };
    // условная инструкция if
    if (user.age >= 12) {
        console.log('Приятного просмотра');
    }
    // инструкция цикла for
    for (i = 1; i < 11; i++) {
        console.log('это ' + i + ' слоник');
    }
</script>
```

Вот действия, которые выполняются данными инструкциями:

- **let i;** – инструкция, которая создаёт переменную с помощью ключевого слова let;
- **const user = { age: 15 };** – инструкция, которая создаёт переменную user с помощью ключевого слова const и присваивает ей объект, содержащий одно свойство - age;
- **if** – инструкция, которая выполняет действия в фигурных скобках, в том случае, если условие истинно;
- **for** – инструкция, которая выполняет код в фигурных скобках заданное количество раз, в данном случае 10 раз.

Инструкции в JavaScript **следует** завершать точкой с запятой. Блоки кода в условных конструкциях и циклах, которые заканчиваются фигурной скобкой **не требуют** после себя точку с запятой.

Выражения

Выражение (англ. *expression*) – это одно из ключевых понятий в JavaScript. Оно представляет собой некоторый фрагмент кода, который всегда **возвращает значение**.

> ***К сведению***. Понятия **Инструкция** (*statement*) и **Выражение** (*expression*) – пригодятся нам при изучении пользовательских **функций**.

Простейшая форма выражения — литерал — нечто, вычисляемое само в себя, например, **число** 100, **строка** "Hellow world". Переменная тоже может быть выражением, так как она вычисляется в присвоенное ей значение.

В свою очередь **выражения JavaScript состоят** из:

- **значений**
- **операторов**
- **операндов**
- **ключевых слов**
- **комментариев**.

Операторы и операнды

Операнды — это данные, обрабатываемые сценарием JavaScript. В качестве операндов могут быть:

- **простые** типы данных,
- **сложные** типы данных,
- другие **выражения**.

В зависимости от **количества операндов** различают следующие типы операторов:

- **унарный** — в операции участвует один операнд;
- **бинарный** — в операции участвуют два операнда;
- **тернарный** — комбинирует три операнда.

Операторы — это символы языка, выполняющие **различные операции** с данными. Операторы могут записываться с помощью символов пунктуации или ключевых слов.

Приведу несколько примеров выражений:

example_2. Примеры выражений

```
<script>
     var name, login;  // выражение 1
     name = "Denis";  // выражение 2
     login = "master";  // выражение 3
     let user = name + " (" + login + ")";  // выражение 4
     console.log (user);  // выражение 5
</script>
```

Некоторые примеры этих выражений имеют **побочные действия**. Т.е. они не только возвращают значение, но выполняют также некоторые дополнительные **действия**.

> ***Таким образом***. **Выражение в JavaScript может быть инструкцией**, но инструкция выражением нет.

Выражение, приведенное в следующем примере, говорит браузеру, что нужно вывести строку "Добро пожаловать на pechora-PROger!" внутри тега <h1> HTML документа:

example_3. Выражение, меняющее свойство узла

```
<!-- html-код -->
```

```
<h2 id="msg"></h2>

<!-- html-код -->

<script>

    document.getElementById("msg").innerText = "Добро пожаловать
    на pechora-PROger!";

</script>
```

Операции в выражениях выполняются **последовательно в соответствии со значением приоритета** (чем больше значение приоритета, тем он выше). **Возвращаемый результат** не всегда имеет значение того же типа, что и тип обрабатываемых данных.

Например, в операциях сравнения участвуют операнды различных типов, но возвращаемый результат всегда будет логического типа (true/false).

Примеры выражений:

- **7 + 3** – выражение, в котором используется оператор +; результатом этого выражения будет число 10;
- **'pechora'** + ' ' + **'PROger'** – результатом этого выражения будет новая строка 'pechora PROger';
- **name = 'Denis'** – выражение, в котором используется оператор присваивания; здесь переменной name присваивается значение 'Denis'; результатом этого выражения будет строка 'Denis';
- **key++** – здесь используется оператор ++, который увеличивает значение переменной на 1; результатом этого выражения будет значение переменной key, увеличенное на 1;
- **2 > 3 || 2 > 1** – выражение, в котором используются несколько операторов; оно возвращает значение true;
- **myFun(3, 5)** – вызов функции также является выражением, т.к. функция всегда возвращает какое-либо значение.
- **console.log('Hello, World!')** – в качестве значения данное выражение возвращает undefined. В этом легко убедиться если его обернуть в ещё одну конструкцию console.log().

Ключевые слова и идентификаторы

Ключевые слова – это зарезервированные слова, которые **являются частью синтаксиса** языка программирования.

Например:

```
let msg = 'hello';
```

В этом примере используется ключевое слово **let**. С помощью него мы можем создавать **переменные** в JavaScript.

Ключевые слова нельзя использовать для обозначения идентификаторов.

В JavaScript очень много ключевых слов. Вот некоторые из них: `await`, `break`, `catch`, `class`, `const`, `continue`, `else`, `export`, `extends`, `for`, `function`, `if`, `let`, `return`, `static`, `this`, `throw`, `try`, `while`.

Идентификатор – это имя, которые мы можем дать переменной, функции, классу и так далее. Составлять имя мы можем из букв, цифр, нижнего подчеркивания и знака доллара.

При этом идентификатор не может начинаться с цифры, а также в качестве него нельзя использовать ключевые слова. Кроме этого, JavaScript чувствителен к регистру.

Комментарии

Комментарии JavaScript – это подсказки, которые программист может добавить, чтобы облегчить чтение и понимание своего кода. Они полностью игнорируются движками JavaScript.

Есть два способа добавить комментарии к коду:

- // – однострочные комментарии;
- /* */ – многострочные комментарии;

Однострочные комментарии

В JavaScript любая строка, начинающаяся с //, является однострочным комментарием. Например:

```
// выведем сообщение в консоль
console.log(`Жизнь прекрасна!`);
```

Однострочный комментарий можно использовать ещё так:

```
console.log(`Мир разумен и справедлив!`); // выведем сообщение в консоль
```

Многострочные комментарии

В JavaScript любой текст между /* и */ является многострочным комментарием. Например:

```
/*
    этот комментарий
    расположен на нескольких строчках
*/
```

Комментарии также очень часть используют для отключения кода **при отладке**. Например, если нам не нужно выполнять какую-то инструкцию, то мы можем её просто закомментировать:

```
myFn1();
// myFn2();
myFn3();
```

Это может очень ценным решением при отладке, потому что вместо удаления некоторого кода мы можем его просто закомментировать.

Арифметические операторы

В JavaScript существует множество различных **операторов**: арифметические операторы, операторы присваивания, строковые операторы, операторы сравнения, логические операторы, операторы типов, побитовые операторы.

> ***Операторы и операнды***.
>
> - **Числа** в арифметической операции называют **операндами**.
> - **Операция**, совершаемая между двумя операндами, называется **оператор**.

Арифметические (математические) **операторы** используют в качестве своих операндов **числа** (или **литералы** или **переменные**) и в качестве результата возвращают одно числовое значение.

Арифметические операторы выполняют математические операции так же, как и большинство других языков программирования.

Например:

```
// деление на ноль возвращает бесконечность Infinity

console.log(1 / 0); /* Infinity */
```

Если один из операндов является строкой, интерпретатор JavaScript попытается преобразовать его в числовой тип, а после выполнить соответствующую операцию. Если преобразование типов окажется невозможным, будет получен результат **NaN** (не число).

Стандартными арифметическими операторами являются:

- сложение (+),
- вычитание (-),
- умножение (*)
- деление (/).

Оператор	Описание
+	**Сложение**. Складывает числовые операнды. Если один из операндов — строка, то результатом выражения будет строка.
-	**Вычитание**. Выполняет вычитание второго операнда из первого.
*	**Умножение**. Умножает два операнда.
/	**Деление**. Делит первый операнд на второй. Результатом деления может являться как целое, так и число с плавающей точкой.
%	**Остаток от деления**. Вычисляет остаток, получаемый при целочисленном делении первого операнда на второй.
**	**Возведение в степень.** Возводит основание в показатель степени.
-	**Унарный минус.** Преобразует положительное число в отрицательное, и наоборот.

Примеры целочисленного деления:

```
console.log(6 % 2); // 0

console.log(11 % 3); // 2

console.log(12 % 12); // 0

console.log(12 % 13); // 12

console.log(12 % 20); // 12
```

example_4. Арифметические операторы

```
<script>

    // пример 1

    let res1 = 8 / 2 + 5 - -18 / 6 * 2;

    // (8 / 2) + 5 - (-18 / 6 * 2);

    console.log(res1); // 15

    // пример 2

    var x = 2;

    let res2 = 8 / 2 + 6 - -9 * x;
```

```
    console.log(res2); // 28

    // пример 3

    var x = 5;

    var y = 8;

    let res3 = x** (3 + 4) / (y + 2) % 2;

    console.log(res3); // 0.5
</script>
```

Операторы присваивания

Операторы присваивания присваивают значения переменным JavaScript.

В результате операции присваивания операнду слева от оператора присваивания (знак "=") устанавливается значение, которое берётся из правого операнда.

Основным оператором присваивания является =, он присваивает значение правого операнда операнду, находящемуся слева. Таким образом, выражение **a = b** означает, что a присваивается значение b.

Существуют также **комбинированные** (составные) **операторы присваивания**, которые используются для сокращённого представления операций, описанных в следующей таблице:

Оператор	Альтернатива	Описание
=	a = b	**Присваивание**. Используется для присваивания значения переменной
+=	a = a + b	**Комбинированный оператор**. Выполняет присваивание с операцией. Между первым и вторым операндом выполняется соответствующая операция, затем результат присваивается первому операнду.
-=	a = a – b	
***=**	a = a * b	
/=	a = a / b	
%=	a = a % b	

Комбинированные операторы используются очень часто.

example_5. Операторы присваивания

```
<script>

	var x = 32;

	x /= 4; // аналогично: x = x / 4, x = 8

	x %= 3; // аналогично: x = x % 3, x = 2

	x *= 7; // аналогично: x = x * 7, x = 14

	x -= 9; // аналогично: x = x - 9, x = 5

	x += 6; // аналогично: x = x + 6, x = 11

	console.log(x); // 11

</script>
```

Операторы инкремента и декремента

Операции **инкремента** и **декремента** являются унарными и производят увеличение и уменьшение значения операнда на единицу. В качестве операнда может быть переменная, элемент массива, свойство объекта. Чаще всего такие операции используются для увеличения счетчика в цикле.

Оператор	Описание
++x	**Префиксный инкремент**. Увеличивает операнд на единицу.
x++	**Постфиксный инкремент**. Прибавляет к операнду единицу, но результатом выражения будет являться первоначальное значение операнда.
--x	**Префиксный декремент.** Уменьшает на единицу операнд, возвращая уменьшенное значение.
x--	**Постфиксный декремент.** Уменьшает на единицу операнд, возвращая первоначальное значение.

Постфиксные операторы:

1. сначала возвращают значение переменной,
2. затем увеличивают или уменьшают значение этой переменной.

Пример:

```
var a = 10;
var b = a++;
console.log(`a=${a}; b=${b}`); // выводит a=11; b=10
```

Сначала переменной b присвоено значение переменной a, а уже затем переменная a была инкрементирована (увеличена на единицу).

Префиксные операторы:

1. увеличивают или уменьшают значение переменной,
2. возвращают значение этой переменной.

Пример:

```
var a = 10;
var b = --a;
console.log(`a=${a}; b=${b}`); // выводит a=9; b=9
```

Сначала переменная a была декрементирована (уменьшена на единицу), а уже затем ее новое значение было присвоено переменной b.

Операции инкремента и декремента на практике применяются очень часто. Например, они встречаются практически в любом цикле for.

example_6. Операторы инкремента и декремента

```
<script>
    let x = 6;
    let y = 19;
    let result = x / 2 + 5 - -(--y) / x++;
    // 3 + 5 - -(18) / 6;  при этом x = 7, y = 18
    console.log("x = ", x); console.log("y = ", y);
    // 8 - (-3)
    console.log(result); // 11
</script>
```

Операторы сравнения

Операторы сравнения используются для сопоставления операндов, результатом выражения может быть одно из двух значений — **true** или **false**. Операндами могут быть не только **числа**, но и **строки**, **логические значения** и **объекты**.

> ***Важно***. **Сравнение может выполняться только для чисел и строк**, поэтому операнды, не являющиеся числами или строками, **преобразуются**.

Вот несколько правил, по которым происходят **преобразования** операндов:

- Если оба операнда не могут быть успешно преобразованы в числа или строки, операторы всегда возвращают **false**.
- Если оба операнда являются **строками**/**числами** или могут быть преобразованы в **строки**/**числа**, они будут сравниваться как **строки**/**числа**.
- Если один операнд является строкой/преобразуется в строку, а другой является числом/преобразуется в число, то оператор попытается преобразовать строку в число и выполнить сравнение чисел. Если строка не является числом, она преобразуется в значение **NaN** и результатом сравнения будет **false**.

Чаще всего операции сравнения используются при организации в программах **ветвлений** (управляющие конструкции – следующая тема курса).

Оператор	Описание
==	**Равенство**. Проверяет две величины на совпадение, допуская преобразование типов. Возвращает **true**, если операнды совпадают, и **false**, если они различны.
!=	**Неравенство**. Возвращает **true**, если операнды не равны
===	**Идентичность**. Проверяет два операнда на «идентичность», руководствуясь строгим определением совпадения. Возвращает **true**, если операнды равны без преобразования типов.
!==	**Неидентичность**. Выполняет проверку идентичности. Возвращает **true**, если операнды не равны без преобразования

	типов.
>	**Больше**. Возвращает **true**, если первый операнд больше второго, в противном случае возвращает **false**.
>=	**Больше или равно**. Возвращает **true**, если первый операнд не меньше второго, в противном случае возвращает **false**.
<	**Меньше**. Возвращает **true**, если первый операнд меньше второго, в противном случае возвращает **false**.
<=	**Меньше или равно**. Возвращает **true**, если первый операнд не больше второго, в противном случае возвращает **false**.

example_7. Операторы сравнения

```
<script>

    console.log(5 == "5"); // true

    console.log(5 === "5"); // false

    console.log(5 == 5.0); // true

    console.log(false === 0); // false

    console.log(1 !== true); // true

    console.log(1 != true); // вернет false, так как true
    преобразуется в 1

    console.log(3 > -3); // true

    console.log(3 >= "2"); // true

</script>
```

Строковые операторы

Существует несколько операторов, которые работают со строками особым образом.

Оператор сложения (+) может использоваться для объединения строк. При использовании со строками, оператор сложения (+) часто называют оператором **конкатенации**.

Оператор	Описание
+	**Конкатенация**. Оператор работает слева направо, выполняя объединение строк. Если первый операнд является строкой, последующие операнды будут преобразованы в строки и далее выполнится их объединение.
+=	**Конкатенация с присваиванием**. Выполняется объединение двух строк и результат присваивается переменной
>, <, >=, <=, ==	**Сравнение**. Строки сравниваются по алфавиту, буквы в верхнем регистре всегда меньше букв в нижнем регистре. Сравнение строк основывается на **номерах символов, указанных в стандарте Unicode**, где прописные буквы идут раньше, чем строчные.

example_8. Строковые операторы

```
<script>

    console.log("1" + "10"); // 110

    console.log("1" + 10); // 110

    console.log("1" > "10"); // false

    console.log("10" == 10); // true

    console.log(7 + 3 + " негритят"); // 10 негритят

    console.log("Карты, деньги, " + 1 + 1 + " ствола"); //
    Карты, деньги, 11 ствола

    console.log("JAVASCRIPT" == "javascript"); // false

    x = "Стив";
```

```
    console.log(x += " Джобс"); // Стив Джобс
</script>
```

Логические операторы

Логические операторы позволяют комбинировать условия, возвращающие **логические величины**. Чаще всего используются в условном выражении **if**.

Оператор	Описание
&&	**Логическое И.** Возвращает **true**, только если оба операнда истинны. При выполнении операции сначала проверяется значение первого операнда. Если оно имеет значение **false**, то значение второго оператора не учитывается и результату выражения присваивается **false**.
||	**Логическое ИЛИ.** Возвращает **true**, если хотя бы один операнд истинен, т.е. проверяет истинность как минимум одного условия.
!	**Логическое НЕ.** Изменяет значение оператора на обратное - с **true** на **false** и наоборот.

Так как логические выражения вычисляются слева направо, они проверяются на возможность выполнения **сокращённой оценки** с использованием следующих правил:

```
// --> сокращение с результатом false
// при операторе && и первом операнде - false, дальнейшие проверки
// бессмысленны
false && expression
// --> сокращение с результатом true
// при операторе || и первом операнде - true, дальнейшие проверки
// бессмысленны
true || expression
```

Правила логики гарантируют, что сокращенная оценка всегда дает корректные вычисления. Обратите внимание, что часть **expression**, представленных выше выражений, не вычисляется.

example_9. Логические операторы

```
<script>
    let a = 3;
    let b = 5;
    let c;
    let count = ((c = a++ - -b) + 1) && (a / c); // 0.5
    // выражение в первых скобках не false, идем дальше
    (операция - &&)
    // результат по выражению (a / c)
    // анализ переменных
    console.log("a=" + a, "b=" + b, "c=" + c, "count=" + count);
    let result = count && b || -7; // 5
    // count не false, идем дальше (операция - &&)
    // b не false - стоп (операция - ||)
    // результат по переменной - b
    console.log(result);
</script>
```

Операторы типов

Оператор	Описание
typeof	Возвращает тип переменной
instanceof	Возвращает **true**, если объект является экземпляром

	определенного объектного типа

Оператор **typeof** — это унарный оператор, который записывается перед своим единственным операндом любого типа и возвращает **строковое** значение, содержащее **тип операнда**.

Операнд оператора typeof может быть записан в двух формах – **в скобках** () и **без скобок**.

example_10. Операторы типов

```
<script>
	console.log(typeof undefined); // undefined
	console.log(typeof true); // boolean
	console.log(typeof (false)); // boolean
	console.log(typeof 5); // number
	console.log(typeof (“”)); // string
	console.log(typeof null); // object
	console.log(typeof {name: “”}); // object
	console.log(typeof function(){}); // function
</script>
```

Задание № 1

Напишите программу, выполняющую перевод 1000 **долларов** ($) в **юани** (¥) через рубли (**Р**). Расчет выполнить исходя из следующего курса:

- 1**$ =** 82,27 **Р**
- 1**Р =** 0,089 **¥**

Результат расчета **вывести** в консоль.

Задание № 2

Напишите программу, которая **вычислит** и **выведет** в консоль результат выражения:

1 + 2 - 3 + 4

где:

1. "5 в степени 3"
2. "12 разделить на 3"
3. остаток от деления "30 на 4"
4. "минус 7 умножить на 2"

Результат каждого действия сохраните в **отдельной** переменной.

Задание № 3

Вам предложен **фрагмент** кода.

```
let x = 6;

let y = 21;
```

1. `let result = --x % 2 + 5 - -(--y) / x++ || 1;`
2. `let result = 1 || --x % 2 + 5 - -(--y) / x++;`

Рассчитайте **значение выражений** без **использования калькулятора**.

Напишите программу, сверьте **вывод** программы с **расчетным** значением. Выведите результат в **консоль**.

Задание № 4

Вам предложен **фрагмент кода**. Рассчитайте вывод **каждой инструкции console.log** без использования калькулятора и прочих подручных средств.

Напишите программу, сверьте **выводы** программы с **расчетными** значениями.

```
<script>
    let x = 5;
    let y = 2;
    let r = --x / y;
    console.log(`
        r = ${r}
        x = ${x}
        y = ${y}
    `);
    let t = (r++ / y) / x--;
    console.log(`
        t = ${t}
        r = ${r}
        x = ${x}
        y = ${y}
    `);
    let s = (x - --y) / t++ + r;
    console.log(`
        s = ${s}
        t = ${t}
        r = ${r}
```

```
            x = ${x}

            y = ${y}

        `);
</script>
```

Задание № 5

Из предложенных блоков **div** файла **div.html** раздаточного материала составьте **массив объектов** JavaScript. Сохраните массив в файле **data.js**. Подключите файл созданного массива к основному файлу программы **index.html**.

Используя приведенный ниже **фрагмент кода** создайте блок и вставьте его в дерево HTML-документа.

```
// var num = ... (*) // место для вставки выражения
let div = document.createElement("div");
div.innerHTML = `
     <h1>Название: ${data[num].title}</h1>
     <img src="images/${data[num].img}">
     <p>Год выпуска: ${data[num].year}</p>
     <p>Страна: ${data[num].country}</p>
     <p>Актеры:</p>
     <ul>
          <li>${data[num].heroes[0]}</li>
          <li>${data[num].heroes[1]}</li>
          <li>${data[num].heroes[2]}</li>
     </ul>
`;
let body = document.body;
body.insertAdjacentElement("afterbegin", div);
```

На место выражения (*) **поочередно** вставляйте выражения, представленные ниже. Прядок вставки выражений следующий:

1. (*) = 0
2. (*) = 1
3. (*) = 2
4. (*) = 3

Выражения для вставки:

- `var num = (4 * 7 % 5 + 1)**1/2;`
- `var num = (5 * 6 - 5)**(1/2) % 6 + -(8/2);`
- `var num = ( (2 / 5) * (1 / 2) * 10 + 1) % 4;`
- `var num = (8**2 - 54) / 2 % 5;`

Тестируйте сценарий каждый раз при вставке **нового** выражения.