**Модуль 1**

Что такое JavaScript?

JavaScript — это не просто язык программирования; это живое воплощение гибкости и эволюции в мире технологий. Давайте разберёмся, что такое JavaScript, зачем он нужен и почему его считают одним из ключевых инструментов современной разработки.

История JavaScript началась в 1995 году, когда Брендан Эйх, работающий в компании Netscape, создал этот язык всего за 10 дней. Да, вы не ослышались — за десять дней! Тогда он назывался LiveScript, но маркетинговый ход, связанный с популярностью Java, заставил переименовать его в JavaScript. Однако общего между ними почти нет. Если Java — это строго типизированный, объектно-ориентированный язык, то JavaScript изначально создавался как динамичный, лёгкий инструмент для взаимодействия с пользователем в веб-браузере.

Поначалу его недооценивали: считали языком для добавления простых «фишек» вроде всплывающих окон или проверки форм. Однако с развитием интернета и технологий JavaScript превратился в нечто большее.

JavaScript — это язык программирования, который работает прямо в браузере. Он позволяет делать веб-страницы интерактивными: анимации, динамическая загрузка данных, реакция на действия пользователя. Сегодня он используется не только в браузерах, но и на серверах, мобильных устройствах, а также в приложениях для настольных ПК.

Это интерпретируемый, высокоуровневый язык с динамической типизацией. Его ключевые особенности:

1. ***Лёгкость освоения****. Даже новички могут написать первые строки кода за считанные минуты.*
2. ***Гибкость****. Отсутствие строгой типизации делает разработку быстрой и удобной.*
3. ***Асинхронность****. Поддержка асинхронного программирования позволяет эффективно работать с большими объёмами данных и сетью.*

Мир программирования невозможно представить без JavaScript. Вот несколько примеров его применения:

1. ***Веб-разработка****. Это главный язык для создания клиентской части сайтов. Благодаря библиотекам и фреймворкам вроде React, Vue и Angular, разработчики создают удобные и быстрые интерфейсы.*
2. ***Серверная разработка****. Node.js позволяет использовать JavaScript на стороне сервера, делая его универсальным инструментом.*
3. ***Мобильные приложения****. React Native, Ionic и другие технологии помогают создавать приложения под iOS и Android.*
4. ***Игры****. С помощью HTML5 и движков (Phaser или Three.js) можно разрабатывать игры прямо в браузере.*
5. ***Облачные технологии и IoT****. JavaScript используется для управления облачными сервисами и устройствами интернета вещей.*

*Как работает JavaScript?*

JavaScript исполняется в браузере через так называемый JavaScript-движок. У каждого браузера свой движок: к примеру, у Chrome это V8, у Firefox — SpiderMonkey. Эти движки компилируют код в машинные инструкции, которые исполняются процессором.

Стоит отметить, что современные браузеры предоставляют **DOM API**, который позволяет JavaScript взаимодействовать с элементами HTML-страницы, изменять их, добавлять события, загружать данные с сервера и многое другое.

JavaScript стал ядром веб-разработки благодаря своей универсальности и активному развитию. Современные фреймворки и инструменты – TypeScript (надстройка над JavaScript) или Webpack, делают разработку удобной, структурированной и масштабируемой.

У языка невероятно активное сообщество, которое создаёт обширную экосистему библиотек, модулей и решений практически для любых задач.

Сегодня JavaScript уже далеко не тот простой язык, который был создан в 1995 году. Благодаря новым стандартам (ECMAScript) он становится всё более мощным и удобным. Сложно представить разработку веб-приложений или интерфейсов без этого языка.

Итак, JavaScript — это не просто язык. Это ключевой инструмент, который связывает разработчиков по всему миру, позволяет создавать интерактивные продукты и служит основой для цифровой трансформации. Его история — это история инноваций и поиска идеального баланса между простотой и мощностью.

Мир технологий меняется быстро, но JavaScript неизменно остаётся его сердцем.

**Запуск JavaScript**

JavaScript код пишется прямо на HTML странице внутри тега script. Этот тег можно размещать в любом месте страницы.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>GPT </title>

<script>

JS код

</script>

</head>

<body>

содержимое страницы

</body>

</html>

Первая программа на JavaScript

# Первая программа на JavaScript

# Первая программа на JavaScript

# Первая программа на JavaScript

# Первая программа на JavaScript

# Первая программа на JavaScript

# Первая программа на JavaScript

# мм Первая программа на JavaScript

# мм Первая программа на JavaScript

# Первая программа на JavaScript

**Первая программа на Java Script**

Напишем нашу первую программу на JavaScript.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>GPT</title>

<script>

alert(‘Привет’);

</script>

</head>

<body>

программирование

</body>

</html>

Скопируйте этот код и разместите в HTML файле. Затем откройте этот файл в браузере - и вы увидите диалоговое окошко с текстом.

**Комментарии**

JavaScript поддерживает два вида комментария: однострочный и многострочный.

**Однострочный комментарий:**

alert(‘Привет’);

**Многострочный комментарий:**

/\*

комментарий

\*/

alert(‘Привет’);

# Переменные в JavaScript

# Переменные в JavaScript

# Переменные в JavaScript

# Переменные в JavaScript

# Переменные в JavaScript

**Переменные в JavaScript**

# Переменные в JavaScript

Основным понятием любого языка программирования является **переменная*.*** **Переменная** -это контейнер, в котором мы можем хранить какие-либо данные, например, строки или числа.

Каждая переменная должна иметь имя, которое может состоять из латинских букв, чисел, символов $ и знаков подчеркивания. При этом первый символ имени переменной не должен быть цифрой.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **str** | **my\_str** | **myStr** | **a1** | **$** | **$a** | **$$a** |

# Использование перемен

# Использование переменных

**Для того, чтобы использовать переменную необходимо**:

* **объявить*:*** написать перед ее именем ключевое слово let.

let А;

* **присвоить ей значение:**

А = 5;

* **вывести значение переменной на экран:**

alert(А);

let А; // объявим переменную

А = 3; // присвоим ей значение

аlert (А); // выведем значение переменной на экран

# Объявление нескольких переменных в JavaScript

# Объявление нескольких переменных в JavaScrip

**Можно объявить несколько переменных:**

let А = 5;

let В = 10;

let С = 15;

let А, В, С; // объявляем все 3 переменные

// Присваиваем переменным значения:

А = 5;

В = 10;

С = 15;

**Можно изменить значение переменной:**

let А; // объявляем переменную

А = 5; // записываем в нее значение 2

alert(А); // выведет 2

А = 5; // записываем теперь значение 2, затирая значение 5

alert(А); // выведет 2

**Одну и ту же переменную нельзя объявить несколько раз через let.**

**Можно просто ввести две разных переменных:**

let А = 1;

alert(a);

let В = 2;

alert(b);

**Типы данных**

**String:** представляет строку

**пример 1:**

let name = "Иван";

let age = 30;

**пример 2:**

let template String = `Привет, меня зовут ${name}, и мне ${age} лет.`;

console.log(templateString); // Выведет: Привет, меня зовут Иван, и мне 30 лет.

**Number:** представляет числовое значение

let integerNumber = 10;

let anotherInteger = -5;

let zero = 0;

console.log(integerNumber); // Выведет: 10

console.log(anotherInteger); // Выведет: -5

console.log(zero); // Выведет: 0

**NaN-** обозначает *не число*. Такое значение получается тогда, когда вы пытаетесь делать математические операции не с числами, а с чем-то другим. К примеру, такое значение даст попытка умножить строку на число:

alert('abc' \* 3); // выведет NaN

Любые операции с NaN приводят к тому, что результатом все равно остается NaN

**Infinity и -Infinity**

Значения Infinity и -Infinity обозначают соответственно бесконечность и минус бесконечность. Такие значения получаются, если какое-то число поделить на ноль - в этом случает JavaScript не выдает ошибку, как в других языках программирования, а возвращает эти значения:

alert( 1 / 0); // выведет Infinity

alert(-1 / 0); // выведет -Infinity

**BigInt**- это встроенный тип данных в JavaScript, который позволяет работать с целыми числами произвольной длины без потери точности.

const bigNum1 = 10000000000000000000n;

const bigNum2 = 20000000000000000000n;

const sum = bigNum1 + bigNum2;

const difference = bigNum2 - bigNum1;

const product = bigNum1 \* bigNum2;

const quotient = bigNum2 / bigNum1; // Результат – BigInt

**вoolean**-логический тип, который может принимать только два значения: **true** или **false,** правда или ложь. Если мы присвоим переменной значением **true**, JS автоматически определит ее тип как boolean.

let isRaining = true;

let isSunny = false;

if (isRaining) {

console.log("Не забудьте взять зонт!");

} else if (isSunny) {

console.log("Отличный день для прогулки!");

} else {

console.log("Погода не определена.");

}

**null** - этот тип данных имеет всего одно значение: null. Которое выражает нулевое или «пустое» значение.

let a = null;

alert(a); // выведет null

**undefined** - тип данных, который обозначает неопределённость.

let a;

alert(a); // выведет undefined

**object** - это ссылочный тип данных, то есть это значение в памяти, на которое возможно сослаться с помощью идентификатора.

let object = new Object({'a': 1, 'b': 2, 'c': 3});

console.log(object);

Результат выполнения кода:

{'a': 1, 'b': 2, 'c': 3}

**symbol** - это уникальный и неизменяемый тип данных, который может быть использован как идентификатор для свойств объектов

let sym1 = Symbol();

let sym2 = Symbol();

console.log(sym1, sym2);

console.log(sym1 === sym2); // false

**Числа**

**Метод Math.round (число)**

**Метод Math. Round** позволяет округлятьзначение до целого числа.

let x = 5.83;

console.log(Math.round(x)); //вернёт: 6

Округление числа производится по математическим правилам. Если после запятой идёт цифра от 0 до 4, то дробная часть просто отбрасывается. А если после запятой идёт цифра от 5 до 9, то дробная часть отбрасывается, а к целой части прибавляется единица.

**Метод toFixed**

Метод toFixed округляет число до заданной точности.

число.toFixed (точность)

Этот метод возвращает не число а строку. Чтобы преобразовать её в число можно добавить + к возвращаемому значению.

let num = x.toFixed(1);

console.log(+num); //вернёт: 5.8

**Метод Math.floor и Math.ceil**

Метод Math.floor() округляет в меньшую сторону. Он отбрасывает дробную часть числа. А метод Math.ceil() округляет в большую сторону. Он отбрасывает дробную часть, а к целой части прибавляет единицу.

console.log(Math.floor(x)); // вернёт 5

console.log(Math.ceil(x)); // вернёт 6

**Метод Math.random**

Метод Math.random() генерирует случайное число в диапазоне от 0 до 1. При этом ровно 1 не бывает, число вегда будет меньше.

Обычно случайное число нужно получить в каком-то конкретном диапазоне.

let rand = -2 + Math.random() \* (5 - (-2));

console.log(rand);

Максимальное будет 4.999999999. Полученные значения можно округлить до нужной точности.

**Метод Math.pow**

Метод Math.pow возводит число в заданную степень. Первым параметром передается число, вторым - в какую степень его возвести.

Возведем число 3 в степень 4:

console.log (Math.pow(3, 4));

Результат выполнения кода:

81

Возведём число 2 в степень -5:

console.log (Math.pow(2, -5));

Результат выполнения кода:

0.03125

Существует оператор \*\*, возводящий числа в степень:

console.log (3 \*\* 4);

Результат выполнения кода:

81

**Метод Math.max**

Метод Math.max

Метод Math.max возвращает максимальное число из группы чисел, переданных в функцию. Если в функцию ничего не передано, то будет возращено -Infinity.

Выведем максимальное число из группы чисел:

console.log(Math.max(1, 5, 10, 34, 100));

Результат выполнения кода:

100

Выведем максимальное число из группы чисел:

console.log(Math.max(-1, 0, -20, -56, -100));

Результат выполнения кода:

0

В следующем примере выведется -Infinity, так как методу не передали параметров:

console.log(Math.max());

Результат выполнения кода:

-Infinity

По умолчанию функция не работает с массивами. Однако ее можно заставить сделать это с помощью [оператора spread](https://code.mu/ru/javascript/book/prime/spread/extreme-array-values/). Выведем максимальное значение массива:

let arr = [1, 5, 10, 34, 100];

let max = Math.max(...arr);

console.log(max);

Результат выполнения кода:

100

Можно заставить функцию работать с массивами с помощью метода [apply](https://code.mu/ru/javascript/manual/context/apply/):

let arr = [1, 5, 10, 34, 100];

let max = Math.max.apply(null, arr);

console.log(max);

Результат выполнения кода:

100

**Метод Math.sqrt**

Метод Math.sqrt возвращает квадратный корень числа.

Возьмем квадратный корень из 4:

console.log(Math.sqrt(4));

Результат выполнения кода:

2

Получим квадратный корень из 15:

console.log(Math.sqrt(15));

Результат выполнения кода:

3.872983346207417

Получим квадратный корень из -100. Так как передано отрицательное число, то будет выведен NaN:

console.log(Math.sqrt(-100));

Результат выполнения кода:

NaN

**Метод Math.min**

Метод Math.min возвращает минимальное число из группы чисел, переданных параметрами. Если параметрами ничего не передано, то будет возращено Infinity.

Выведем минимальное число из группы чисел:

console.log(Math.min(40, 20, 42, 100, 67));

Результат выполнения кода:

20

Выведем минимальное число из группы чисел:

console.log(Math.min(-1, -100, -30, -25, 40));

Результат выполнения кода:

-100

В следующем примере выведется Infinity, так как методу не передали параметров:

console.log(Math.min());

Результат выполнения кода:

Infinity

**Пример 4**

По умолчанию функция не работает с массивами. Однако ее можно заставить сделать это с помощью [оператора spread](https://code.mu/ru/javascript/book/prime/spread/extreme-array-values/). Выведем минимальное значение массива:

let arr = [1, 5, 10, 34, 100];

let min = Math.min(...arr);

console.log(min);

Результат выполнения кода:

1

**Пример 5**

Можно заставить функцию работать с массивами с помощью метода [apply](https://code.mu/ru/javascript/manual/context/apply/):

let arr = [1, 5, 10, 34, 100];

let min = Math.min.apply(null, arr);

console.log(min);

Результат выполнения кода:

1

**Метод Math.abs**

Метод Math.abs возвращает модуль числа, то есть из отрицательного числа делает положительное.

Выведем модуль числа -3:

console.log(Math.abs(-3));

Результат выполнения кода:

3

Выведем модуль числа 3:

console.log(Math.abs(3));

Результат выполнения кода:

3

**Метод Math.sin**

Метод Math.sin возвращает синус числа (это число должно быть углом в радианах).

Выведем синус числа 7:

console.log(Math.sin(7));

Результат выполнения кода:

0.6569865987187891

**Метод Math.cos**

Метод Math.cos возвращает косинус числа (это число должно быть углом в радианах).

Math.cos(число);

Выведем косинус числа 34:

console.log(Math.cos(34));

Результат выполнения кода:

-0.8485702747846052

**Метод Math.sin**

Метод Math.sin возвращает синус числа (это число должно быть углом в радианах).

Выведем синус числа 7:

console.log(Math.sin(7));

Результат выполнения кода:

0.6569865987187891

**Метод Math.asin**

Метод Math.asin возвращает арксинус числа. Параметр метода должен находиться в промежутке от -1 до 1, иначе будет возращен NaN.

Выведем арксинус числа 0.45.

console.log(Math.asin(0.45));

Результат выполнения кода:

0.4667653390472964

Выведем арксинус числа -1:

console.log(Math.asin(-1));

Результат выполнения кода:

-1.5707963267948966

Выведем арксинус числа 45. Так как параметр больше 1, то будет возращен NaN:

console.log(Math.asin(45));

Результат выполнения кода:

NaN

**Метод Math.atan**

Метод Math.atan возвращает арктангенс числа. Параметр метода должен находиться в промежутке от -Пи/2 и Пи/2, иначе будет возращен NaN.

Выведем арктангенс числа 2:

console.log(Math.atan(2));

Результат выполнения кода:

1.1071487177940904

**Метод Math.tan**

Метод Math.tan возвращает тангенс числа (это число должно быть углом в радианах).

Выведем тангенс 89:

console.log(Math.tan(89));

Результат выполнения кода:

1.6858253705060158

Метод Math.acos

**Метод Math.acos**

Метод Math.acos возвращает арккосинус числа. Параметр метода должен находиться в промежутке от -1 до 1, иначе будет возращен NaN.

Давайте выведем арккосинус числа 0.34:

console.log(Math.acos(0.37));

Результат выполнения кода:

1.191787306098946

Выведем арккосинус числа 1:

console.log(Math.acos(1));

Результат выполнения кода:

0

В следующем примере выведется NaN, так как параметр больше 1:

console.log(Math.acos(3));

Результат выполнения кода:

NaN

**Метод toExponential**

Метод toExponential переводит число в экспоненциальную нотацию. Необязательный параметр - число цифр после запятой, если его нет, то будет выведено столько цифр после запятой, сколько нужно для записи.

В данном примере была выведена экспоненциальная нотация числа 5:

let num = 5;

console.log(num.toExponential());

Результат выполнения кода:

5e+0

В данном примере была выведена экспоненциальная нотация числа 5, причем с 3 знаками после запятой:

let num = 5;

console.log(num.toExponential(3));

Результат выполнения кода:

5.000e+0

**Свойство Math.PI**

Свойство Math.PI возвращает число Пи с 15-ю знаками после запятой.

Выведем значение числа Пи:

console.log(Math.PI);

Результат выполнения кода:

3.141592653589793

Свойство Math.E

**Свойство Math.E**

Свойство Math.E содержит в себе экспоненту с точностью до определенного знака после запятой.

Выведем на экран значение данной константы:

console.log(Math.E);

Результат выполнения кода:

2.718281828459045

**Математические операции с числами**

**Сложение:**

let А= 1 + 2;

alert(А); // выведет 3

**Вычитание:**

let В = 3 - 2;

alert(В); // выведет 1

**Умножение:**

let С = 3 \* 2;

alert(c); // выведет 6

**Деление:**

let D= 4 / 2;

alert(d); // выведет 2

**Возведение в степень:**

2 \*\* 2 // 4

let А = 2

А\*\* = 4 // 16

**Остаток от деления**

Оператор взятия остатка %.

аlert ( 5 % 2 ); // 1, остаток от деления 5 на 2

аlert( 8 % 3); // 2, остаток от деления 8 на 3

аlert( 8 % 4); // 0, остаток от деления 8 на 4

**Дробные числа**

Целая и дробная части отделяются друг от друга точкой.

let a = 0.5;

alert(a); // выведет 0.5

let a = 0.5 + 0.5;

alert(a); // выведет 1

**Инкремент и декремент**

Инкремент увеличивает значение переменной на 1. Декремент уменьшает значение переменной на 1.

Инкремент: ++

Декремент: --

let num = 0;

num++; // прибавляем к переменной a число 1

alert(num); // выведет 1

let num = 0;

num--; // отнимаем от переменной num число 1

alert(num); // выведет -1

**Отрицательные и положительные числа**

Перед отрицательным числом ставится знак минус

let a = -1;

alert(a); // выведет -1

Знак минус можно писать как к числам, так и к переменным:

let a = 1;

let b = -a; // записали в b содержимое a с обратным знаком

alert(b); // выведет -1

Перед положительным числом ставится знак плюс

let a = +1;

alert(a); // выведет 1

**Математические операции с переменными**

**Математические операции можно производить не только над числами, но и над переменными.**

let А= 2;

let В = 3;

alert(А + В); // выведет 5

**Приоритет математических операций**

**В JavaScript в начале выполняется умножение и деление, а потом уже сложение и вычитание.**

Пример: в начале 5 умножится на 2 и затем к результату прибавится 1

let А = 5 \* 2 + 1;

alert(А); // выведет 11 (результат 10 + 1)

**Умножение и деление имеют равный приоритет и выполняются по очереди слева направо.**

Пример: вначале выполнится деление, а потом умножение

let А = 6 / 2 \* 4;

alert(a); // выведет 12 (результат 3\* 4)

**В следующем примере каждая новая операция деления будет применяться к предыдущей:**

let А = 16 / 2 / 2 / 2;

alert(А); // выведет 2

**Можно указывать приоритет операций с помощью круглых скобок.**

let А = 2 \* (2 + 3);

alert(А); // выведет 10 (результат 2 \* 5)

**Примечание: скобок может быть любое количество, в том числе и вложенных друг в друга:**

let А = 2 \* (2 + 4 \* (3 + 1));

alert(А);by

**Строки**

Для применения строк применяют кавычки: одинарные ('), двойные (") или обратные (`)

let str = 'abc';

alert(str); // выведет 'abc'

let str = "abc";

alert(str); // выведет 'abc'

**Получение символа в строке.**

Каждый символ в строке имеет свой порядковый номер: первый символ номер 0, второй символ - номер 1, третий символ номер 2 и так далее.

Нумерация символов начинается с нуля. Можно получить доступ к определенному символу строки по его номеру. Для этого пишется имя переменной, после этого имени ставятся квадратные скобки и в этих скобках указывается номер символа.

let str = 'abcde';

alert(str[0]); // выведет 'a'

alert(str[1]); // выведет 'b'

alert(str[2]); // выведет 'c'

Номер символа также может хранится и в переменной:

let str = 'abcde';

let num = 3; // номер символа в переменной

alert(str[num]); // выведет 'd'

**Получение последнего символа в строке**

let str = 'abcde';

let last = str.length - 1; // номер последнего символа

alert(str[str.length - 1]); // выведет 'e'

Последний символ строки в JavaScript

**Длина строки**

Количество символов в строке содержится в свойстве length:

let str = 'abcde';

alert(str.length); // выведет 5

Свойство length можно применять непосредственно к строке:

alert('abcde'.length); // выведет 5

**Шаблонные строки**

Существует специальный тип кавычек – обратные (косые):

let str = `abc`;

alert(str); // выведет 'abc'

В косых кавычках можно выполнять вставку переменных. Для этого имя переменной нужно написать в конструкции ${}.

let str = 'xxx';

let txt = 'aaa ' + str + ' bbb';

Этот код можно переписать следующим образом:

let str = 'xxx';

let txt = `aaa ${str} bbb`;

**Сложение строк в JavaScript**

Для сложения строк используется оператор +:

let str = 'abc' + 'def'; // складываем две строки

alert(str); // выведет 'abcdef'

let str1 = 'abc';

let str2 = 'def';

alert(str1 + str2); // выведет 'abcdef'

**Многострочность**

В JavaScript строках, созданных через одинарные или двойные кавычки, не допустим перенос строки.

Косые кавычки специально предназначены для создания многострочных строк:

let str = `abc

def`; // так будет работать

**Преобразование строки в верхний регистр**

**Метод toUpperCase**

let str = 'abcde';

console.log(str.toUpperCase());

Результат выполнения кода:

'ABCDE'

**Метод** [**slice**](https://code.mu/ru/javascript/manual/string/slice/)

let str = 'abcde';

let res = str.slice(0, 1).toUpperCase() + str.slice(1);

console.log(res);

Результат выполнения кода:

'Abcde'

**Преобразование строки в нижний регистр**

**Метод toLowerCase**

let str = 'ABCDE';

console.log(str.toLowerCase());

Результат выполнения кода:

'abcde'

**Возвращение символа в строке**

**Метод charAt**

let str = 'abcde';

let char = str.charAt(0);

console.log(char);

Результат выполнения кода:

'a'

**Условия**

**Условные операторы**

# Конструкция if if-else

# if-else

**Конструкция if-else**

Конструкция **if** задает условие. С помощью этой конструкции мы можем задать программе вопрос "больше ли переменная **a,** чем переменная **b**", и в зависимости от ответа выполнить или один код, или другой.

Как правило данная конструкция используется в комбинации с конструкцией [**else**](https://code.mu/ru/javascript/manual/lang/else/)**.**

**Синтаксис**

if (логическое выражение) {

/\*

расположенный здесь код выполниться,

если логическое выражение верное

\*/

};

Если в фигурный скобках только одно выражение, можно эти фигурные скобки не писать.

**Операторы больше и меньше**

Для проверки того, которое из значений больше, а которое меньше, используются операторы:

> больше

 >= больше или равно

 < меньше

<= меньше или равно

**Проверим, значение переменной больше нуля или нет:**

let test = 1;

if (test > 0) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Значение переменной отрицательное число:**

let test = -1;

if (test > 0) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

**Значение переменной равно 0. В этом случае мы попадем в блок else, так как в нашем условии написано, что переменная test должна быть строго больше нуля:**

let test = 0;

if (test > 0) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

**Значение переменной больше или равно:**

let test = 0;

if (test >= 0) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Значение переменной меньше 0:**

let test = 0;

if (test < 0) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

**Значение переменной меньше или равно:**

let test = 0;

if (test <= 0) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Проверка на равенство**

Для проверки двух значений на равенство используется оператор ==.

let test = 10;

if (test == 10) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Проверка на неравенство**

Оператор != проверяет на неравенство. Давайте для примера проверим, что переменная test не равна 0:

let test = 1;

if (test != 0) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

Поменяем значение переменной:

let test = 0;

if (test != 0) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

**Сравнение переменных**

let test1 = 1;

let test2 = 2;

if (test2 > test1) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Равенство строк**

При использовании конструкции if можно сравнивать и строки. Для примера пусть в переменной test хранится какая-то строка, например 'abc':

let test = 'abc';

Давайте проверим, равно ли содержимое переменной test строке 'abc':

let test = 'abc';

if (test == 'abc') {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Равенство строк и чисел**

**Сравним строку '3' и число 3:**

if ('3' == 3) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Равенство по значению и типу**

Вместо оператора == следует использовать оператор ===.

При сравнении строки '3' и числа 3 условие будет ложным, так как переменные, хотя и равны по значению, но не равны по типу:

if ('3' === 3) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

При сравнении двух строк '3' условие будет истинно:

if ('3' === '3') {

console.log('+++'); // сработает это

} else {

console.log('---');

}

Как и при сравнении чисел:

if (3 === 3) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

Разница между оператором == и оператором === проявляется именно тогда, когда значения одинаковые, но разный тип данных. В остальных случаях эти операторы работают одинаково. К примеру, при сравнении разных чисел, конечно же, выведется '---':

if (2 === 3) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

**Неравенство по значению и типу**

Кроме оператора != существует также оператор !==, учитывающий тип при сравнении. Давайте рассмотрим различия между ними на примерах.

Пусть с помощью оператора != сравниваются два числа 3. Данный оператор сравнивает значения на то, что они НЕ равны. Так как наши значения как раз-таки равны, то условие будет ложным:

if (3 != 3) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

Пусть теперь одно из наших значений будет в кавычках. В этом случае оператор != все равно посчитает их равными (так как совпадает значение, а тип не важен для данного оператора) и опять условие будет ложным:

if ('3' != 3) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

Давайте теперь сравним два числа 3 с помощью оператора !==. Он также посчитает их равными:

if (3 !== 3) {

console.log('+++');

} else {

console.log('---'); // сработает это

}

А вот если теперь одну из троек взять в кавычки, то оператор !== посчитает наши тройки неравными, так как, хотя значения их и совпадают, но у них разный тип:

if ('3' !== 3) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Логическое И**

Можно группировать операции сравнения с помощью оператора &&, представляющего собой логическое И.

Если переменная num больше нуля и *одновременно* меньше 10, только тогда выведется '+++':

let num = 3;

if (num > 0 && num < 10) {

console.log('+++');

} else {

console.log('---');

}

Условия могут налагаться не на одну переменную, а на разные.

Если переменная num1 равна 2 и одновременно переменная num2 равна 3, только тогда условие будет истинно:

let num1 = 2;

let num2 = 3;

if (num1 == 2 && num2 == 3) {

console.log('+++');

} else {

console.log('---');

}

**Логическое ИЛИ**

Оператор || представляет собой логическое ИЛИ и требует выполнения хотя бы одного условия.

Если переменная num1 больше 0 или переменная num2 больше 0, то условие будет истинно (достаточно выполнения хотя бы одного из условий):

let num1 = 10;

let num2 = -5;

if (num1 > 0 || num2 > 0) {

console.log('+++'); // сработает это

} else {

console.log('---');

}

**Условия с булевыми значениями**

Пусть у нас есть какая-то переменная, которая может принимать значения true или false:

let test = true;

Давайте напишем if, проверяющий нашу переменную на значение true:

let test = true;

if (test === true) {

console.log('+++');

} else {

console.log('---');

}

**Вложенные конструкции** **if-else**

Конструкции **if-else** можно вкладывать друг в друга произвольным образом.

let num = 3;

if (num >= 0) {

if (num <= 5) {

console.log('меньше или равно 5');

} else {

console.log('больше 5');

}

} else {

console.log('меньше нуля');

}

**Конструкции switch-case**

С помощью конструкции switch-case можно выполнять различный код, в зависимости от значения переменной.

let num = 1;

switch (num) {

case 1:

console.log('value1');

break;

case 2:

console.log('value2');

break;

case 3:

console.log('value3');

break;

default:

console.log('incorrect value');

break;

}

**Массивы**

Массив представляет собой переменную, в которой в упорядоченном виде можно хранить целый набор каких-то значений.

Значения, которые хранит массив, называются *элементами*. Элементы разделяются между собой запятой. После этой запятой принято ставить пробелы.

Получение элементов в массиве

let arr = ['a', 'b', 'c'];

console.log(arr[0]); // выведет 'a'

console.log(arr[1]); // выведет 'b'

console.log(arr[2]); // выведет 'c'

Обращение к элементам массива осуществляется подобно обращениям к символам строки. Каждый элемент массива имеет свой номер: первый элемент - номер 0, второй - номер 1 и так далее. Эти номера называются *ключами* элементов массива.

Длина массива

Длина массива находится с помощью свойства length:

let arr = ['a', 'b', 'c'];

console.log(arr.length) // выведет 3

Последний элемент в массиве

let arr = ['a', 'b', 'c'];

console.log(arr[arr.length - 1]) // выведет 'c'

Изменение элементов массива

Элементы массивов можно изменять:

let arr = ['a', 'b', 'c'];

arr[0] = '!';

console.log(arr); // выведет ['!', 'b', 'c']

Перезапись элементов массива

Можно также прочитывать текущее значение элемента, выполнять с ним какие-то операции и записывать измененное значение обратно в этот элемент:

let arr = ['a', 'b', 'c'];

arr[0] = arr[0] + '!';

arr[1] = arr[1] + '!';

arr[2] = arr[2] + '!';

console.log(arr); // выведет ['a!', 'b!', 'c!']

Предыдущий код можно переписать через оператор +=:

let arr = ['a', 'b', 'c'];

arr[0] += '!';

arr[1] += '!';

arr[2] += '!';

console.log(arr); // выведет ['a!', 'b!', 'c!']

Инкремент и декремент элементов массива

let arr = [1, 2, 3, 4];

arr[0]++;

++arr[1];

arr[2]--;

--arr[3];

console.log(arr); // выведет [2, 3, 2, 3]

Добавление элементов по ключам

let arr = [];

arr[0] = 'a';

arr[1] = 'b';

arr[2] = 'c';

console.log(arr); // выведет ['a', 'b', 'c']

Можно также добавлять элементы в массив, уже заполненный данными:

let arr = ['a', 'b', 'c'];

arr[3] = 'd';

console.log(arr); // выведет ['a', 'b', 'c', 'd']

Разреженные массивы

Если в массиве будут ключи с пропусками, то можно получить *разреженный* массив.

let arr = ['a', 'b', 'c'];

Добавим к этому массиву еще один элемент так, чтобы ключи имели пропуски:

arr[4] = '!';

В результате в массиве образуется дыра, имеющая значение *undefined*:

console.log(arr); // выведет ['a', 'b', 'c', undefined,   
 '!']

console.log(arr.length); // выведет 5

Ключи массивов из переменных

let arr = ['a', 'b', 'c'];

let arr = ['a', 'b', 'c'];

console.log(arr[0]); // выведет 'a'

Давайте теперь ключ выводимого элемента не будем писать непосредственно в квадратных скобках, а запишем его в переменную:

let arr = ['a', 'b', 'c'];

let key = 0; // запишем ключ в переменную

Используем теперь нашу переменную для вывода соответствующего элемента:

let arr = ['a', 'b', 'c'];

let key = 0; // запишем ключ в переменную

console.log(arr[key]); // выведет 'a'

**Циклы**

Циклы используются для того, чтобы некоторый участок кода выполнился несколько раз подряд.

Циклы могут повторять некоторый код заданное количество раз. Каждый такой проход цикла называется итерацией цикла.

В циклах часто используются специальные переменные, которые каждую итерацию увеличивают свое значение на единицу. Такие переменные называются счетчиками циклов. Счетчики используются для того, чтобы подсчитывать, сколько раз выполнился цикл. Для счетчиков принято использовать буквы i, j и k.

**Перебор массива циклом for-of**

Цикл for-of позволяет последовательно перебирать элементы массивов. Синтаксис

for (let переменная для элемента of массив) {

/\*

в переменную для элемента по очереди

будут попадать элементы перебираемого массива.

\*/

}

let arr = [1, 2, 3, 4, 5];

for (let elem of arr) {

console.log(elem);

}

**Перебор объекта циклом for-in**

Для перебора объектов предназначен цикл for-in.

Синтаксис

for (let переменная для ключа in объект) {

}

let obj = {a: 1, b: 2, c: 3};

for (let key in obj) {

console.log(key); // выведет 'a', 'b', 'c'

}

Выведем элементы:

for (let key in obj) {

console.log(obj[key]); // выведет 1, 2, 3

}

**Цикл while**

Цикл while будет выполняться до тех пор, пока *истинно* выражение, переданное ему параметром. Он позволяет выполнять произвольное количество итераций.

Синтаксис

while ( пока выражение истинно ) {

выполняем этот код циклически;

в начале каждого цикла проверяем выражение   
 в круглых скобках

}

Цикл закончится, когда выражение перестанет быть истинным.

Выведем с помощью цикла while числа от одного до пяти:

let i = 1;

while (i <= 5) {

console.log(i);

i++;

}

Поделим заданное число на 2 столько раз, пока результат не станет меньше 10:

let num = 500;

while (num > 10) {

num = num / 2;

}

console.log(num); // результат

**Цикл do..while**

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, цикл повторяется.

let i = 1;

do{

    console.log(i);

    i++;

}while(i <= 5)

**Цикл for**

Цикл for позволяет повторить некоторый код заданное количество раз.

Синтаксис

for (начальные команды; условие окончания; команды после   
 прохода) {

тело цикла

}

Выведем с помощью цикла for выведем последовательно числа от 1 до 9:

for (let i = 1; i <= 9; i++) {

console.log(i); // выведет 1, 2... 9

}

Увеличим счетчик не на 1, а на 2:

for (let i = 1; i <= 9; i += 2) {

console.log(i); // выведет 1, 3, 5...

}

Выполним обратный отсчет:

for (let i = 10; i > 0; i--) {

console.log(i); // выведет 10, 9, 8...

}

Переберём массив циклом for.

let arr = [1, 2, 3, 4, 5];

Выведем элементы массива в цикле:

for (let i = 0; i <= arr.length - 1; i++) {

console.log(arr[i]);

}

Выведем элементы в обратном порядке:

for (let i = arr.length - 1; i >= 0; i--) {

console.log(arr[i]);

}

**Инструкция break и continue**

Инструкция break позволяет досрочно завершать цикл.

for(let i=1; i <= 6; i++){

     if(i===4) break;

     console.log(i);

}

console.log("Конец работы");

Согласно условию цикла блок цикла должен выполняться 6 раз, произвести 6 итераций. Однако поскольку в блоке цикла происходит поверка if (i===4) break;, то, когда значение переменной i достигнет 4, то данное условие прервет выполнение цикла с помощью оператора break. И цикл заершит работу.

Существует также инструкция continue, запускающая новую итерацию цикла.

for(let i=1; i <= 6; i++){

    if(i===4) continue;

    console.log(i);

}

console.log("Конец работы");

В этом случае, когда значение переменная i станет равной 4 , то выражение i===4 возвратит true, поэтому будет выполняться конструкция if(i===4) continue;. С помощью оператора continue она завершит текущую итерацию, далее идущие инструкции цикла не будут выполняться.