**Конспект «Основы программирования на JavaScript»**

Программа — это набор команд. JavaScript выполняет программу последовательно, команда за командой. Команды разделяются точкой с запятой ;.

**Консоль**

Чтобы вывести информацию в консоль, используем команду console.log:

console.log(*данные для вывода в консоль*);

Эту команду можно использовать в любом месте программы и выводить в консоль результаты выполнения операций и текстовые подсказки. Текстовые подсказки, в отличие от результатов операций, нужно заключать в кавычки.

**Комментарии**

Комментарии не выводятся в консоль и не влияют на работу программы, но видны разработчику. Код внутри комментариев не выполняется. Обычно в них пишут поясняющие тексты для себя, или для других, или для себя в будущем.

Комментарии бывают двух типов: однострочные и многострочные:

// Эта строка кода не выполнится. Однострочный комментарий.

/\*

Все эти строки кода не выполнятся.

Так как это многострочный комментарий.

\*/

**Типы данных**

С разными типами данных можно производить разные действия, поэтому программисту важно знать, с чем он работает. В нашей консоли тип данных выводится в скобках, например (String) или (Number).

Существуют простые и сложные типы данных. Простые:

* number — числа: целые и с точкой;
* string — строки;
* boolean — логические, или булевы, значения: true — «истина» и false — «ложь»;
* undefined — «не определено», англ.

Строки нужно оборачивать в кавычки: одинарные или двойные.

Сложные, или составные, типы содержат не одно, а несколько значений. Массив, array, хранит последовательность значений, и порядок этих значений важен. Объект, object, состоит из множества пар «ключ-значение», порядок этих пар не важен.

// Массив

[1, 2, 3, 4, 5]

// Объект

{month: 'june', day: 15}

**Переменные**

Переменная — просто название для данных, которое можно делать понятным для людей. Переменные упрощают работу с памятью: они «приклеиваются» к ячейкам памяти, как наклейка с названием приклеивается к папке с документами.

В JavaScript переменные можно создавать командой let, за которой следует имя переменной:

let имяПеременной;

Имя переменной можно записать по-разному. Два самых популярных способа: camelCase (верблюжья нотация) и snake\_case (змеиная нотация). В первом случае все слова пишутся слитно и каждое слово, за исключением первого, начинается с большой буквы (myNumber, userName). Во втором случае все слова разделяются нижним подчёркиванием (my\_number, my\_name).

Имена переменных в JavaScript чувствительны к регистру: myname и myName — две разные переменные. Имя переменной может содержать буквы, цифры и знак подчёркивания, но оно не должно начинаться с цифры. Кроме того, в качестве имени переменной нельзя использовать ключевые слова, такие как let или if. Вот [полный список](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Lexical_grammar#%D0%9A%D0%BB%D1%8E%D1%87%D0%B5%D0%B2%D1%8B%D0%B5_%D1%81%D0%BB%D0%BE%D0%B2%D0%B0) этих ключевых слов.

После создания переменной её можно использовать в других командах, например, выводить в консоль:

// Обратите внимание, что кавычек нет!

console.log(имяПеременной);

Если обратиться к пустой переменной, то получим undefined — «не определено». Чтобы записать в переменную данные, ей их нужно *присвоить*. Для операции присваивания используется знак равенства:

let timeInHours; // Объявляем переменную

console.log(timeInHours); // Выведет: undefined

timeInHours = 2; // Присваиваем одно значение

console.log(timeInHours); // Выведет: 2

timeInHours = 'три часа'; // Присваиваем совершенно другое значение

console.log(timeInHours); // Выведет: три часа

Команда let для создания каждой переменной используется всего один раз. Дальше мы обращаемся к переменной по её имени, без let. Если повторно задать значение переменной, то значение этой переменной изменится. Предыдущее значение при этом исчезнет. Это называется переопределением переменной.

**Операции и операторы**

Команды состоят из операций. 5 + 10; — это операция. Она состоит из оператора, +, и двух операндов, 5 и 10.

Оператор указывает, что произойдёт с операндами. Операции бывают унарными, бинарными и тернарными, в зависимости от количества операндов. Бинарные операции самые распространённые.

Над разными типами операндов можно производить разные операции, поэтому важно понимать, данные какого типа хранятся в переменных.

Порядок выполнения операций зависит от их приоритета. Если у операций одинаковый приоритет, они выполняются слева направо. Приоритет различных операторов можно посмотреть [здесь](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Operator_Precedence).

**Арифметические операции**

Арифметические операции в JavaScript выполняются так же, как в математике: сначала умножение, потом сложение. Изменить порядок операций можно с помощью круглых скобок. Снова как в математике: выражение в скобках посчитается в первую очередь.

|  |  |
| --- | --- |
| Сложение | + |
| Вычитание | - |
| Умножение | \* |
| Деление | / |

**Конкатенация**

Самая частая строковая операция — это «склеивание» строк, или конкатенация:

let name = 'Кекс';

'Инструктор' + 'Кекс'; // Результат: 'ИнструкторКекс'

'Инструктор ' + 'Кекс'; // Результат: 'Инструктор Кекс'

'Инструктор ' + name; // Результат: 'Инструктор Кекс'

Конкатенация позволяет делать сообщения программ более информативными и «человечными».

**Приведение типов**

Что будет, если использовать операнды разного типа?

'Время, мин: ' + 50; // Результат: 'Время, мин: 50'

'2' \* 50; // Результат: 100

JavaScript попытается привести операнды к одному типу и выполнить операцию. Подходящий тип будет выбираться в зависимости от операции.

Плюс может быть знаком сложения или конкатенации, но так как один из операндов — строка, то сложение не подходит. Поэтому число 50 приводится к строке '50' и склеивается со строкой 'Время, мин: '.

Звёздочка — это знак умножения, со строками она не используется. Поэтому JavaScript пытается превратить строку '2' в число, и ему это удаётся. Затем числа 2 и 50 перемножаются, и получается 100.

Из-за того, что JavaScript умеет изменять тип операндов на лету, он называется языком со *слабой типизацией*.

# Конспект «Условия»

## Линейные и нелинейные программы

Линейные программы всегда выполняют *одни и те же команды*.

Нелинейная программа выполняет *разные команды* в зависимости от *разных условий*. Нелинейные программы ещё называют программами с ветвлением, а команды, которые выполняются в зависимости от условий, — ветками.

## if и else

Чтобы программа проверяла условия и принимала решения на основе результатов проверок, используют оператор if:

if (условие) {

действия;

}

Здесь «условие» — это выражение, возвращающее true или false, а «действия» внутри фигурных скобок — это команды, которые выполняются, если условие удовлетворено. Удовлетворённым считается условие, которое возвращает true.

Чтобы создать ветку, которая будет срабатывать, если условие не выполнено, используем else:

if (условие) {

действия;

} else {

другие действия;

}

Ветка «действия» срабатывает, если условие выполнено. Ветка «другие действия» срабатывает, если условие не выполнено. Такие конструкции можно читать так: *если условие выполняется, сделай действие, иначе сделай другие действия*.

## Вложенные условия

Что делать, если принятие решения зависит не от одного, а от двух и более условий? Можно использовать один if, а затем, внутри ветки, выполнить ещё одну проверку.

if (условие1) {

if (условие2) {

действия;

}

}

Затем внутри вложенного условия можно добавить ещё одно и так до бесконечности.

Вложенные условия могут сделать код сложным и запутанным. Если вложенность большая, то понять, почему выполняется то или иное действие, становится трудно.

## Операторы сравнения

Для сравнения значений используют операторы «больше» >, «меньше» <, «больше или равно» >= и «меньше или равно» <=.

console.log(1 > 1); // Выведет: false

console.log(1 < 1); // Выведет: false

console.log(1 >= 1); // Выведет: true

console.log(1 <= 1); // Выведет: true

Операторы сравнения работают не только с числами, но и с другими типами данных.

Для сравнения строк JavaScript использует таблицу кодирования [Unicode](https://home.unicode.org/" \t "_blank). Порядок символов в ней совпадает с порядком букв в алфавите. Чем больше порядковый номер символа в таблице, тем больше символ. Обратите внимание, строчные буквы в таблице Unicode идут после заглавных, поэтому они считаются «больше»:

console.log('Б' > 'А'); // Выведет: true

console.log('а' > 'А'); // Выведет: true

Строки JavaScript сравнивает посимвольно. Если первый символ в первой строке больше первого символа во второй строке, то считается, что первая строка больше. Если первые символы совпадают, то сравниваются вторые символы и так далее. Если все символы совпадают, но одна из строк длиннее, то она и считается большей. Например:

console.log('Кот' > 'Код'); // Выведет: true

console.log('JavaScript' > 'Java'); // Выведет: true

Если сравниваются данные разных типов, то они приводятся к числу. При этом false становится нулём, а true — единицей.

console.log(2 > '1'); // Выведет: true

console.log(false <= 0); // Выведет: true

console.log(true >= 1); // Выведет: true

## Равенство и неравенство

В JavaScript можно также проверить значения на равенство и неравенство. При этом используют операторы ==, !=, === и !==.

|  |  |  |
| --- | --- | --- |
| Оператор | Название | Действие |
| == | Нестрогое равенство (с приведением типов) | Сравнивает два значения, перед этим приводит одно из значений к типу другого. Если значения равны, возвращает true. |
| === | Строгое равенство (без приведения типов) | Сравнивает два значения. Если типы значений разные или значения не равны, возвращает false. |
| != | Неравенство (с приведением типов) | Сравнивает два значения, перед этим приводит одно из значений к типу другого. Если значения не равны, возвращает true. |
| !== | Строгое неравенство (без приведения типов) | Сравнивает два значения. Если типы значений разные или значения не равны, возвращает true. |

## String и Number

Можно привести числовое значение к строковому типу. Один из способов — использовать команду String:

String(число);

Чтобы превратить строку в число, используют команду Number:

Number(строка);

## Приведение к логическому типу

В условии все значения приводятся к логическому типу. Поэтому мы можем использовать в качестве условий любые значения: числа, строки, true и false, а также переменные, которые содержат такие данные.

Все числа, кроме нуля, — true, при этом 0 — false. Все строки, кроме пустой строки, — true, пустая строка '' — false. Можно сказать, что значения, которые как бы ничего в себе не содержат (как 0 или пустая строка ''), приводятся к false, а все остальные приводятся к true.

if ('какая-то строка') {

// Непустая строка приводится к true

// Условие выполнится

};

if ('') {

// Пустая строка приводится к false

// Условие не выполнится

};

if (123) {

// Число приводится к true

// Условие выполнится

};

if (0) {

// 0 приводится к false

// Условие не выполнится

};

## Логические операторы

### **Отрицание**

Чтобы создать проверки с отрицанием, используют унарный (с одним операндом) логический оператор !:

let condition = false;

if (!condition) {

// код выполнится

}

### **И и ИЛИ**

Можно комбинировать условия внутри if с помощью логических операторов: «логического И», &&, и «логического ИЛИ», ||.

Оператор «Логическое И», возвращает true только в том случае, если *оба* условия, слева и справа от него, возвращают true.

true && true; // Результат: true

true && false; // Результат: false

false && true; // Результат: false

false && false; // Результат: false

Оператор «логическое ИЛИ», возвращает true если *любое* из условий слева или справа от него, возвращают true.

true || true; // Результат: true

true || false; // Результат: true

false || true; // Результат: true

false || false; // Результат: false

Например:

let conditionOne = true;

let conditionTwo = true;

let conditionThree = false;

let conditionFour = true;

if (conditionOne && conditionTwo) {

// код выполнится

}

if (conditionThree || conditionFour) {

// код тоже выполнится

}

Логические операторы можно комбинировать:

let conditionOne = true;

let conditionTwo = true;

let conditionThree = false;

if (conditionOne && conditionTwo && !conditionThree) {

// код выполнится

}