**let и const против var**

Чтобы объявить переменную, пишут ключевое слово let и имя новой переменной:

Скопировать кодJAVASCRIPT

let championOfTheWorld = 'Франция';

let olympicChampion;

olympicChampion = 'Великобритания';

Это отличный способ определять переменные, но он существовал не всегда: до выхода стандарта ES6 использовали ключевое слово var. Его можно встретить в старом коде:

Скопировать кодJAVASCRIPT

var championOfTheWorld = 'Франция';

var olympicChampion;

olympicChampion = 'Великобритания';

Определение переменных через let не допускает некоторых неочевидных эффектов. Например, использование переменной до её определения ключевым словом var:

Скопировать кодJAVASCRIPT

console.log(x); *// undefined - переменная не определена, но ошибки нет*

var x = 10;

Программистам неудобно читать такой код, поэтому let не позволяет использовать переменную до её явного определения:

Скопировать кодJAVASCRIPT

console.log(y); *// Uncaught ReferenceError: y is not defined - ошибка!*

let y = 10;

Сначала вы должны определить переменную через let, а уже потом использовать её для решения задач.

Всегда пишите let, но помните про var: он может встретиться в старых проектах.

Ещё один способ определения переменных, появившийся с выходом ES6 — const.

const — ключевое слово для создания константных переменных. Если попробовать переопределить константу, движок выдаст ошибку:

Скопировать кодJAVASCRIPT

const championOfTheWorld = 'Франция';

let olympicChampion;

olympicChampion = 'Великобритания';

*// попробуем изменить константу*

championOfTheWorld = 'Хорватия'; *// TypeError: Assignment to constant variable*

const накладывает запрет именно на перезапись. Если объявить через const массив, вы сможете добавлять и удалять его элементы, но переопределить массив целиком не получится:

Скопировать кодJAVASCRIPT

const sisters = ['Вера', 'Надежда'];

sisters[2] = 'Любовь';

console.log(sisters); *// ['Вера', 'Надежда', 'Любовь'] — сработало!*

sisters = ['Трус', 'Балбес', 'Бывалый']; *// Консоль выдаст ошибку: «VM51:1 Uncaught TypeError: Assignment to constant variable. at <anonymous>:1:9»*

Чем строже код — тем лучше. В этом примере мы объявили массив декларацией const. Так следует делать всегда, если вы не планируете перезаписывать значение переменной.

Старайтесь использовать константные переменные const везде, где возможно. В случае, когда значение нужно изменять, используйте let.

Теперь вы знаете, что на бумаге возможности JavaScript определяет стандарт, а в реальности — движок. Разные движки пишут разные люди: отсюда проблемы совместимости.

Вооружимся этими знаниями, начнём глубокое погружение в JavaScript.

# Методы поиска в строке

В прошлых темах вы узнали о том, что у строки есть длина, и научились получать символ строки по индексу. А сейчас поиграем в «Поле Чудес» — будем искать символы в строке. Роль ведущего достаётся JavaScript, роли игроков — за нами.

## Поиск по строке. Метод indexOf

Метод indexOf ищет символ в строке и возвращает его индекс:

Скопировать кодJAVASCRIPT

*/\* Метод indexOf \*/*

'Яндекс.Практикум'.indexOf('Я'); *// 0*

*/\* Если таких символов в строке несколько,*

*метод вернёт индекс первого из них: \*/*

'эспрессо'.indexOf('c'); *// 1*

*/\* Если символа в строке нет, indexOf вернёт -1: \*/*

'эспрессо'.indexOf('к'); *// -1*

indexOf может найти и комбинацию символов. При этом вернётся индекс первого символа из последовательности:

Скопировать кодJAVASCRIPT

const blok = 'Ночь, улица, фонарь, аптека';

blok.indexOf('фонарь'); *// 13*

indexOf чувствителен к регистру:

Скопировать кодJAVASCRIPT

'Шалаш'.indexOf('Ш'); *// 0*

'Шалаш'.indexOf('ш'); *// 4*

## Поиск по строке. Метод includes

Часто нужно проверить, есть ли в строке набор символов:

Скопировать кодJAVASCRIPT

'хлеб да соль'.indexOf('соль') !== -1; *// true*

'хлеб да соль'.indexOf('сахар') !== -1; *// false*

Получается два действия — сначала запускаем indexOf, а потом сравниваем с -1. Это работает, но выглядит не элегантно.

В более свежем стандарте придумали метод includes. Он выполняет проверку, которую мы только что делали вызовом indexOf:

Скопировать кодJAVASCRIPT

'Гарри Поттер и узник Азкабана'.includes('Гарри Поттер'); *// true*

## Поиск в начале и конце строки. Методы startsWith и endsWith

Методы startsWith и endsWith — более узконаправленные альтернативы indexOf.

Метод startsWith сравнивает начало строки с переданной ему подстрокой. Он отвечает на вопрос: «Эта строка начинается с вот такого набора символов?» Если ответ «да», метод возвращает true, иначе — false:

Скопировать кодJAVASCRIPT

'Вендетта'.startsWith('В'); *// true*

'Родина'.startsWith('Картинка в букваре'); *// false*

Метод endsWith делает всё то же самое, только с концом строки. Он отвечает на вопрос: «Эта строка заканчивается вот таким набором символов?»:

Скопировать кодJAVASCRIPT

const theRealEnd = 'Это ещё не конец';

theRealEnd.endsWith('конец'); *// true*

Теперь ваши возможности для работы со строками расширились: вы можете находить в строке символы и их наборы, проверять отдельно начало и конец строки.

# Методы преобразования строк

Есть ещё несколько полезных методов, которые стоит знать, но трудно объединить по какому-то признаку.

## Управление регистром. Методы toLowerCase и toUpperCase

Методы toLowerCase toUpperCase возвращают новую строку, все символы которой переведены в нижний и верхний регистр соответственно:

Скопировать кодJAVASCRIPT

'Включите Caps Lock'.toLowerCase(); *// "включите caps lock"*

'Выключите Caps Lock'.toUpperCase(); *// "ВЫКЛЮЧИТЕ CAPS LOCK"*

Полезно вызывать один из методов, когда нужно сравнить две строки без учёта регистра:

Скопировать кодJAVASCRIPT

const firstStr = 'Такая проверка не пройдёт';

const secondStr = 'таКая пРовеРка не пройдёт';

console.log(firstStr === secondStr); *// false*

console.log(firstStr.toLowerCase() === secondStr.toLowerCase()); *// true*

## Превращение строки в массив. Метод split

Метод split принимает на вход один аргумент — разделитель. Он показывает, где заканчивается один элемент массива и начинается следующий.

Разделителем может быть любой символ или набор символов:

Скопировать кодJAVASCRIPT

'Пришёл. Увидел. Победил.'.split(' '); *// ["Пришёл.", "Увидел.", "Победил."]*

'Пришёл. Увидел. Победил.'.split('. '); *// ["Пришёл", "Увидел", "Победил."]*

## Извлечение части строки. Метод slice

Метод slice возвращает часть строки. На вход он принимает два аргумента — с какого индекса начинать отсчёт и на каком закончить:

Скопировать кодJAVASCRIPT

'Не прислоняться'.slice(6, 10); *// "слон"*

Второй аргумент необязательный: если его не указать, slice() вернёт все символы от заданного в первом аргументе до конца строки:

Скопировать кодJAVASCRIPT

'Яндекс.Практикум'.slice(7); *// "Практикум"*

'Яндекс.Телефон'.slice(7); *// "Телефон"*

'Яндекс.Еда'.slice(7); *// "Еда"*

Со строками разобрались. В следующем уроке посмотрим, что можно делать с числами.

# Методы для работы с числами

## Объект Math и его методы

У объекта Math много методов для работы с числами, основные:

Скопировать кодJAVASCRIPT

*// округляет переданное число «вниз»*

Math.floor(9.99); *// 9*

*// округляет «вверх»*

Math.ceil(9.01); *// 10*

*// округляет до ближайшего целого*

Math.round(9.51); *// 10*

*// возвращает наибольшее из переданных чисел*

Math.max(1, 2, 3, 4, 5); *// 5*

*// возвращает наименьшее из переданных чисел*

Math.min(1, 2, 3, 4, 5); *// 1*

*// возвращает случайное число от 0 включительно до 1 не включительно*

Math.random(); *// 0.31764219954126016*

Если среди этих методов нет нужного вам, загляните [в документацию](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Math). Вполне вероятно, что он найдётся там.

## Работа с дробной частью. Функция parseInt

В поликлинике всех спрашивают: «Сколько вам полных лет?» Чтобы ответить на этот вопрос, нужно взять свой возраст (например, 37 лет, 8 месяцев и 10 дней) и отбросить всё, кроме количества лет (37).

Так эта логика выглядит в JavaScript:

Скопировать кодJAVASCRIPT

let age = '37 лет, 8 месяцев и 10 дней';

console.log(parseInt(age)); *// 37*

Функция parseInt приводит переданный аргумент к целому числу. Она читает аргумент слева направо, если встречает не цифру, останавливается и возвращает всё, что прочитала до этого:

Скопировать кодJAVASCRIPT

parseInt('38 попугаев'); *// 38*

Если первый символ передаваемой строки не цифра, parseInt вернёт NaN:

Скопировать кодJAVASCRIPT

parseInt('Метро 2033'); *// NaN*

Второй аргумент parseInt — система счисления, в которой число передаётся функции. parseInt определяет только то, в какой системе счисления передан первый аргумент, но на выходе мы всё равно получим десятичное число.

Скопировать кодJAVASCRIPT

parseInt('100', 10); *// 100*

parseInt('100', 2); *// 4 (100 в двоичной системе)*

Чаще всего вам нужна десятичная система. Явно указывайте её, чтобы избежать неожиданных результатов.

## Функция parseFloat

Функция parseFloat работает аналогично parseInt, только выделяет не целое число, а число с плавающей точкой:

Скопировать кодJAVASCRIPT

parseFloat('36.6'); *// 36.6*

parseFloat('36.6 нормальная температура человека '); *// 36.6*

## Проверка на принадлежность к целым числам. Метод Number.isInteger

Метод Number.isInteger принимает число как аргумент и проверяет, целое оно или дробное:

Скопировать кодJAVASCRIPT

const eightAndAHalf = 8.5;

Number.isInteger(eightAndAHalf); *// false*

Number.isInteger(Math.floor(eightAndAHalf)); *// true*

# Неявное преобразование типов

JavaScript — язык со слабой типизацией:

* данные могут менять свой тип, даже если разработчик об этом не просил;
* невозможно сложить строку с числом, а две строки можно.

Перед вами пример с конкатенацией из прошлого спринта:

Скопировать кодJAVASCRIPT

console.log(100 + '500'); *// "100500"*

При сложении числа со строкой движок превратит число в строку и только потом выполнит конкатенацию.

При попытке сложить строку и число многие языки программирования начинают «ругаться» и показывать ошибки. Но JavaScript такое прощает. Это накладывает дополнительную ответственность на разработчика: нужно следить за поведением переменных. Иначе можно не заметить незапланированное преобразование типа:

Скопировать кодJAVASCRIPT

const a = 30; *// Количество отжиманий в первом подходе*

const b = 10; *// Количество отжиманий во втором подходе*

console.log('За сегодня вы отжались: ' + a + b + ' раз');

*// "За сегодня вы отжались 3010 раз"*

Неявное преобразование типа — то, которое происходит без прямого указания разработчика. В JavaScript три вида неявного преобразования типов для примитивных значений:

* приведение к строке,
* приведение к числу,
* приведение к логическому типу.

## Приведение к строке

Неявное преобразование к строчному типу происходит при сложении со строкой:

Скопировать кодJAVASCRIPT

1 + ''; *// "1"*

undefined + ''; *// "undefined"*

## Приведение к числу

Движок JavaScript приводит данные разных типов к числам, а затем их сравнивает:

Скопировать кодJAVASCRIPT

null >= 1; *// false*

'451' < 452; *// true*

Оператор «унарный плюс» приводит нечисловые значения после «+» к численному типу. Так +'33' вернёт число 33, +'-77' вернет число -77. Если операторы сложения и унарный плюс стоят рядом, они не помешают друг другу:

Скопировать кодJAVASCRIPT

console.log(67 + +'33'); *// 100*

Приведение к числу выполнит и любой другой арифметический оператор. Кроме сложения со строкой — оно будет воспринято как конкатенация:

Скопировать кодJAVASCRIPT

console.log('3' - 1); *// 2 — выполнилось приведение к числу*

console.log('4' / 4); *// 1 — тоже привелось к числу*

console.log('2' \* 2); *// 4 — опять привелось*

console.log('4' + 4); *// 44 — а здесь выполнится конкатенация*

## Приведение к булевым значениям

В круглых скобках условия if любые данные всегда приводятся к логическому типу:

Скопировать кодJAVASCRIPT

const usernameElement = document.querySelector('.username');

if (usernameElement) {

console.log('Привет, ' + usernameElement.textContent);

}

Если элемента с классом username на странице нет, в переменную usernameElement запишется null. В круглых скобках условной конструкции null приводится к логическому типу и становится false. Поэтому тело условия if выполнено не будет.

# Явное преобразование типов

О неявном преобразовании важно знать, но намеренно использовать его не стоит. Другие разработчики могут неправильно трактовать такой код или просто не заметить неявное преобразование. Более того, вы и сами можете о нём забыть.

Это ведёт к ошибкам в программе, которые очень сложно отловить: они не всегда отображаются в консоли.

Разберёмся, как явно приводить один тип к другому.

## Приведение к строке

Глобальный объект String превращает переданный ему аргумент в строку:

Скопировать кодJAVASCRIPT

const numberToString = String(2); *// "2"*

const nanToString = String(NaN); *// "NaN"*

const undefinedToString = String(undefined); *// "undefined"*

const nullToString = String(null); *// "null"*

const booleanToString = String(true); *// "true"*

## Приведение к числу

Глобальный объект Number приводит переданный ему аргумент к числовому типу:

Скопировать кодJAVASCRIPT

const stringToNumber = Number('2'); *// 2*

const nullToNumber = Number(null); *// 0*

Если передать произвольную строку или undefined, вернётся NaN.

Скопировать кодJAVASCRIPT

const anotherStringToNumber = Number('счастье не за горами'); *// NaN*

const undefinedToNumber = Number(undefined); *// NaN*

## Приведение к булевым значениям

Метод Boolean преобразует переданный ему аргумент к «булю»:

Скопировать кодJAVASCRIPT

Boolean(2) *// true*

Boolean(0) *// false*

Boolean('') *// false*

Boolean('Непустая строка'); *// true*

Разберёмся подробнее, как разные значения приводятся к логическому типу. В JavaScript значения условно делятся на truthy (англ. «правдивые») и falsy (англ. «ложные»). Правдивые значения при приведении типа становятся Истиной, а ложные — Ложью. Правила такие:

* все непустые строки — truthy,
* пустая строка ('') — falsy,
* все ненулевые числа — truthy,
* ноль — falsy,
* NaN, null и undefined — falsy,
* объекты, массивы и функции — truthy.

Скопировать кодJAVASCRIPT

Boolean('Непустая строка'); *// true*

Boolean(''); *// false*

Boolean(1); *// true*

Boolean(0); *// false*

Boolean(NaN); *// false*

Boolean(null); *//false*

Boolean(undefined); *// false*

Boolean({}); *// true*

Boolean([]); *//true*

Запоминать и заучивать это не нужно, но можно воспользоваться такой подсказкой. Всё интуитивно пустое (ноль, NaN, null, undefined, пустая строка) приводится к false. Всё остальное — к true.

**Логические операторы**

Условия позволяют делать в коде «вилки»: выполнять при истинности какого-то условия один код, а в противном случае — другой. Этого достаточно в простейших ситуациях, но обстоятельства бывают разные.

Допустим, вы покупаете машину на вторичном рынке и для вас важны условия:

* пробег не больше 100 тысяч км,
* 1–2 владельца по ПТС,
* не битая,
* не дороже миллиона.

Попробуем написать функцию, которая принимает на вход параметры автомобиля и возвращает true, если машина подходит, и false — если нет:

Скопировать кодJAVASCRIPT

function checkTheCar (distance, ownersNumber, crushed, cost) {

if (distance <= 100000) { *// проверяем пробег*

if (ownersNumber === 1) { *// 1 владелец по ПТС?*

if (crushed === false) { *// не битая?*

if (cost <= 1000000) { *// стоит меньше миллиона?*

return true;

} else return false;

} else return false;

}

else if (ownersNumber === 2) { *// Владелец не 1? Тогда может 2?*

if (crushed === false) { *// не битая?*

if (cost <= 1000000) { *// стоит меньше миллиона?*

return true;

} else return false;

} else return false

} else return false;

} else return false;

}

checkTheCar(50000, 2, false, 700000); *// true. Надо же! Работает.*

Хоть проверки и простые, код выглядит монструозно. Ситуацию исправят логические операторы: они объединят несколько условий в одно.

Логических операторов всего три:

* ! — логическое НЕ,
* || — логическое ИЛИ,
* && — логическое И.

Пойдём по порядку и в следующем уроке разберём логическое НЕ.

# Оператор логическое НЕ (!)

**Логическое НЕ** делает из условия обратное: превращает true в false и наоборот. Для этого перед условием ставят восклицательный знак !:

Скопировать кодJAVASCRIPT

!false; *// true*

Было условие «если три больше двух» — станет «если три НЕ больше двух»:

Скопировать кодJAVASCRIPT

!(3 > 2) === 3 <= 2; *// true.*

*/\* Потому что это превращается в проверку false === false \*/*

Напишем функцию проверки пароля:

Скопировать кодJAVASCRIPT

let password = 'JavaScript';

function checkPassword(pass) {

if (!(pass === password)) {

console.log('Неверный пароль');

return;

}

console.log('Пароль правильный');

}

checkPassword('java script'); *// 'Неверный пароль'*

checkPassword('JavaScript'); *// 'Пароль правильный'*

Если поставить ! перед небулевым значением, движок JS сначала приведёт тип к булю, а затем изменит значение на противоположное:

Скопировать кодJAVASCRIPT

!'Непустая строка' *// false*

Двойное отрицание сработает как перевод значения в логический тип:

Скопировать кодJAVASCRIPT

!!true; *// true*

!!'непустая строка'; *// true*

!!''; *// false*

!!1; *// true*

!!0; *// false*

Оператором НЕ можно создавать анти-условия и приводить данные к булю. В следующих уроках мы расскажем о логических операторах И и ИЛИ.

# Оператор логическое ИЛИ (||)

**Логическое ИЛИ** служит для связи таких условий, из которых должно быть выполнено хотя бы одно. Для этого между простыми условиями ставят два прямых слеша ||:

Скопировать кодJAVASCRIPT

true || false || false; *// true*

*/\* Оператору ИЛИ достаточно, чтобы хотя бы*

*одно простое условие было истинным \*/*

В примере с покупкой автомобиля вы рассматривали варианты машин с 1–2 хозяевами по ПТС. Перепишем условие:

Скопировать кодJAVASCRIPT

(ownersNumber === 1) || (ownersNumber === 2)

Оператор ИЛИ идёт по простым условиям слева направо и проверяет каждое. Когда он встречает правдивое значение, то возвращает его как результат всего условия. Посмотрим на примере:

Скопировать кодJAVASCRIPT

let condition = 0 || NaN || 'строка' || false;

*/\* Значение переменной condition: 'строка'. \*/*

В переменную condition записывается результат — значение первого истинного условия.

## Значение по умолчанию

Часто оператором ИЛИ присваивают переменной значение по умолчанию:

Скопировать кодJAVASCRIPT

function howDoYouDo(answer) {

const result = answer || 'да ничего';

return result;

}

howDoYouDo('всё прекрасно'); *// "всё прекрасно"*

howDoYouDo(); *// "да ничего"*

Пока вы не передаёте аргумент функции howDoYouDo, её вызов возвращает установленное по умолчанию значение 'да ничего'. Это происходит из-за того, что оператор ИЛИ из пустоты и чего-нибудь определённого выбирает определённое.

# Оператор логическое И (&&)

**Логическим И** объединяют условия, которые должны выполняться одновременно. Для этого между условиями ставят двойной амперсанд &&:

Скопировать кодJAVASCRIPT

(distance <= 100000) && (ownersNumber === 1)

Если поставить && между правдивыми условиями, оператор вернёт последнее из них:

Скопировать кодJAVASCRIPT

2 \* 2 === 4 && 5 < 6 && 'Каждый может стать' *// 'Каждый может стать'*

Если среди условий есть ложные, && вернёт из них первое ложное:

Скопировать кодJAVASCRIPT

2 \* 2 === 4 && undefined && 'Каждый может стать' *// undefined*

## Приоритетность операторов

В математике умножение и деление выполняется раньше сложения и вычитания, но действия в скобках всегда совершаются первыми.

С логическими операторами ситуация схожая: сначала выполняется логическое НЕ, потом И, а затем — ИЛИ:

Скопировать кодJAVASCRIPT

const optimism = !'Жить' && !'Быть' || 'Жить и быть';

console.log(optimism); *// "Жить и быть"*

*/\* сначала выполняется !, потом &&, затем || \*/*

Сначала !'Жить' → false, потом !'Быть' → false, затем false && false → false и наконец false || 'Жить и быть' → 'Жить и быть'.

Операции в скобках выполняются первыми:

Скопировать кодJAVASCRIPT

const pessimism = !'Жить' && (!'Быть' || 'Жить и быть');

console.log(pessimism); *// false*

Результат уже другой. Следите за порядком операций и скобками.

## Пример с автомобилем

В исходном примере получился такой код:

Скопировать кодJAVASCRIPT

function checkTheCar (distance, ownersNumber, crushed, cost) {

if (distance <= 100000) { *// проверяем пробег*

if (ownersNumber === 1) { *// 1 владелец по ПТС?*

if (crushed === false) { *// не битая?*

if (cost <= 1000000) { *// стоит меньше миллиона?*

return true;

} else return false;

} else return false;

}

else if (ownersNumber === 2) { *// владелец не 1? Тогда, может, 2?*

if (crushed === false) { *// не битая?*

if (cost <= 1000000) { *// стоит меньше миллиона?*

return true;

} else return false;

} else return false

} else return false;

} else return false;

}

checkTheCar(50000, 2, false, 700000); *// true. Надо же! Работает.*

С логическими операторами вы можете переписать этот кошмар.

Условия покупки машины:

* пробег не больше 100 тысяч км,
* один или два владельца по ПТС,
* не битая,
* не дороже миллиона.

Скопировать кодJAVASCRIPT

function checkTheCar (distance, ownersNumber, crushed, price) {

const condition = distance <= 100000 && (ownersNumber === 1 || ownersNumber === 2) && !crushed && price <= 1000000;

return condition;

}

checkTheCar(50000, 2, false, 700000); *// true — фух, работает*

Разница очевидна. Благодаря логическим операторам код стал понятнее и короче.