**Alert, консоль, подключение JS к странице**

Код на JavaScript можно писать двумя способами:

* между тегов <script></script>, которые мы рекомендуем помещать прямо перед закрывающим </body>;
* в отдельном файле, ссылаясь на него в HTML-документе конструкцией:

Скопировать кодHTML

<script src="путь к файлу"></script>

Очень похоже на подключение CSS, но нужно запомнить:

1. Тегу <script> всегда нужен закрывающий </script>, в отличие от <link>.
2. Путь к JavaScript-файлу задаётся атрибутом src (как с картинками), а не href (как с гиперссылками).

**Уведомить пользователя. alert**

Команда alert выводит в браузере системное окно с текстом, заданным в скобках:

Скопировать кодJAVASCRIPT

alert('Семь раз отмерь - один отрежь.');

Это самый простой способ сказать что-то пользователю, но в современной веб-разработке такие окна применяются редко.

**Вывод информации в консоль. console.log**

Консоль облегчает разработку и помогает найти ошибки в коде страницы.

Чтобы найти консоль в браузере, кликните по окну просмотра правой кнопкой мыши, выберите «Исследовать элемент» и в открывшейся панели вкладку ”Console“. А в нашем тренажёре консоль уже открыта снизу справа.

Для вывода сообщения в консоль воспользуйтесь командой console.log. Например:

Скопировать кодJAVASCRIPT

console.log('Семеро одного не ждут');

Пословицу не увидит обычный пользователь, но увидит разработчик, у которого открыта консоль.

В круглых скобках можно писать не только слова, но и выражения. В консоль попадёт их результат:

Скопировать кодJAVASCRIPT

console.log(2 + 3); *// выведет в консоль результат вычисления: 5*

# Числа, операции с числами

Код на JavaScript хранит данные и манипулирует ими. Например, считает возраст на основе введённой пользователем даты рождения.

Данные бывают разных типов. Первый и самый простой — числа. Манипулировать числами позволяют арифметические операторы:

Скопировать кодJAVASCRIPT

12 + 13 *// оператор сложения*

44 - 4 *// оператор вычитания*

19 \* 3 *// оператор умножения*

57 / 3 *// оператор деления*

То, с чем работают операторы, — в данном случае числа — называют «операнд**»**. Говорят: «левый операнд» и «правый операнд».

Когда в одной строке кода несколько операторов, то операторы \* и / выполняются первыми. Их приоритет выше, чем - и +:

Скопировать кодJAVASCRIPT

console.log(24 / 24 - 12); *// -11*

**Скобки, или оператор группировки,** имеют высший приоритет. Операции в скобках выполняются первыми:

Скопировать кодJAVASCRIPT

console.log(24 / (24 - 12)); *// 2*

**Оператор возведения в степень** \*\*. Основание (слева) умножается на себя число раз, равное степени (справа):

Скопировать кодJAVASCRIPT

console.log(2 \*\* 4); *// 16*

**Оператор взятия остатка, или modulo** %. Возвращает не проценты и не модуль, а остаток от деления левого числа на правое:

Скопировать кодJAVASCRIPT

console.log(5 % 1); *// 0*

console.log(3 % 2); *// 1*

console.log(11 % 4); *// 3*

Заметили, что в конце каждой строки мы ставим точку с запятой? Это необязательно, но в JavaScript так принято.

В длинной программе точка с запятой может уберечь вас от некоторых ошибок, поэтому привычка ставить их полезна.

# Строки, конкатенация, приведение числа к строке

В предыдущем уроке вы выводили в консоль числа. Но в консоль также выводят текстовые сообщения. Текст относится к типу данных, который называют строками. Любой набор символов становится строкой, когда ограничен с двух сторон двойными или одинарными кавычками:

Скопировать кодJAVASCARIPT

console.log("слово");

console.log('Не просто слово, а целое предложение.');

console.log('1984'); // число в кавычках — это строка

В строку могут войти буквы, цифры и любые символы, закодированные по принятому в интернете стандарту «Юникод». Строку начинают и оканчивают одинаковыми кавычками:

Скопировать кодJAVASCARIPT

console.log('текст строки');

console.log('текст строки"); // это ошибка

Если строка содержит кавычки или апострофы, проще всего выбрать другие пограничные символы. Иначе в строку войдёт не всё. Например, правильно вот так:

Скопировать кодJAVASCARIPT

console.log("Республика Кот-д'Ивуар");

console.log('Никогда не говори "никогда" — можешь ошибиться.');

Хоть строку можно заключать и в одинарные, и в двойные кавычки, лучше придерживаться одного стандарта — ставить либо только одинарные, либо только двойные. Поэтому во всех примерах кода, где это возможно, мы будем использовать одинарные кавычки.

## Конкатенация

Оператор сложения склеивает строки:

Скопировать кодJAVASCARIPT

console.log('вино' + 'град'); // "виноград"

Такое сложение разработчики называют «конкатенация». Это не арифметическое действие, а сцепление символов строки и той сущности, с которой строка соединяется.

При сложении строки с числом получается строка:

Скопировать кодJAVASCARIPT

console.log(100 + 500); // сложили два числа, получили число 600

console.log(100 + '500'); // сложили число и строку, получили строку '100500'

## Шаблонные строки

Еще один вид строк — шаблонные строки. Они ограничиваются с обеих сторон обратными кавычками ```. Шаблонные строки могут содержать переносы строки:

Скопировать кодJAVASCARIPT

console.log(`Это - шаблонная строка,

и она может занимать

несколько строк`);

Также шаблонные строки могут содержать вычисляемые выражения внутри фигурных скобок со знаком доллара в начале ${...}:

Скопировать кодJAVASCARIPT

console.log(`${2 + 5} раз отмерь - ${9 - 8} отрежь`); // "7 раз отмерь - 1 отрежь"

# Переменные, типы переменных

Программирование начинается с переменных. Они позволяют запоминать результат промежуточных действий, а это даёт возможность создавать более сложные программы. У переменной есть имя и значение.

Сначала переменную объявляют: ставят JavaScript в известность, что данное имя принадлежит переменной. Это делают ключевым словом let:

Скопировать кодJAVASCRIPT

let champion;

Так объявляют переменную по имени champion.

Переменные именуют так, чтобы по имени было ясно, какие она хранит данные: sum для суммы, username для имени пользователя, productId для идентификатора товара в интернет-магазине. Давайте переменным имена на английском. Если имя состоит из нескольких слов, пробелы не ставьте — используйте camelCase.

Объявить переменную — создать коробку с этикеткой. Это достаточно сделать один раз, поэтому ключевое слово let нужно только при объявлении. А присвоить переменной значение — значит что-то в эту коробку положить. Можно записать в переменную значение при объявлении, а можно — после него. Если записывать значение в переменную после объявления, ключевое слово let писать не нужно:

Скопировать кодJAVASCRIPT

let championOfTheWorld = 'Франция'; *// можно сразу, со словом let*

let olympicChampion;

olympicChampion = 'Великобритания'; *// можно позже, тогда без let*

olympicChampion = 'Бразилия'; *// значение переменной можно изменить*

console.log(olympicChampion); *// Бразилия*

Старое значение из переменной пропадает — чемпион бывает только один.

Необязательно присваивать переменной точное значение, можно указать выражение, в котором это значение вычисляется. Тогда в переменной сохраняется результат расчёта:

Скопировать кодJAVASCRIPT

let twoByTwo = 2 \* 2;

console.log(twoByTwo); *// 4*

Переменные используют и в арифметических операциях, и в сравнении, и в конкатенации, что весьма удобно для вывода результатов вычисления с пояснениями:

Скопировать кодJAVASCRIPT

let hours = 19; *// Текущее время в часах*

*// Устанавливаем в магазине прогрессивную вечернюю скидку:*

*// С 6 до 9 часов вечера скидка растёт на 10% каждый час*

let discount = 0;

if (hours >= 18 && hours <= 21) {

discount = (hours - 17) \* 10;

}

*// Выводим сообщение о текущей скидке*

console.log('После ' + hours + ':00 скидка в магазине ' + discount + '%');

*// "После 19:00 скидка в магазине 20%"*

## var VS let VS const

## var: ограничена областью видимости функции её значение будет undefined если вы попытаетесь обратиться к ней до её объявления.

## let: ограничена областью видимости блока вы получите ReferenceError если попытаетесь обратиться к ней до её объявления.

## const: ограничена областью видимости блока вы получите ReferenceError если попытаетесь обратиться к ней до её объявления. не может быть перезаписана

## Короткая запись

Если переменную нужно поделить, умножить или изменить на известное число, а затем перезаписать новое значение в эту же переменную, есть две формы записи:

Скопировать кодJAVASCRIPT

*// вариант 1*

let year1 = 365;

year1 = year1 + 1; *// 366*

*// вариант 2*

let year2 = 365;

year2 += 1; *// 366 — этот код делает то же самое*

+= — оператор добавочного присваивания, или присваивания со сложением. В JavaScript есть аналогичные операторы для других арифметических действий:

Скопировать кодJAVASCRIPT

let num1 = 10;

num1 += 1; *// 11*

let num2 = 10;

num2 -= 1; *// 9*

let num3 = 10;

num3 \*= 2; *// 20*

let num4 = 10;

num4 /= 5; *// 2*

## Это интересно

В фигурных скобках внутри шаблонной строки ${...} можно писать любые выражения — в том числе содержащие переменные. Это позволяет использовать шаблонные строки вместо конкатенации:

Скопировать кодJAVASCRIPT

*// Выводим сообщение о текущей скидке*

console.log(`После ${hours}:00 скидка в магазине ${discount}%`);

*// "После 19:00 скидка в магазине 20%"*

# Условия, больше-меньше, булевы операции

## Сравнение: больше, меньше, равно

В логических вычислениях возможны два ответа: true (утверждение истинно) или false (утверждение ложно). Самая простая логическая операция — сравнение. Сравнение в JavaScript производят операторами:

Скопировать кодJAVASCRIPT

console.log(7 > 6); *// true — больше*

console.log(7 < 6); *// false — меньше*

console.log(7 >= 6); *// true — больше или равно*

console.log(7 <= 6); *// false — меньше или равно*

console.log(7 === 6); *// false — проверка на равенство*

console.log(7 !== 6); *// true — проверка на неравенство*

Возможно, первые четыре оператора знакомы вам из школьной алгебры. А последние два — проверка на равенство и неравенство — что-то новое, но не волнуйтесь, в них нет ничего сложного. Восклицательный знак в операторе неравенства позаимствован из логической алгебры. Он означает отрицание.

Сравнивать можно не только числа, а вообще любые типы:

Скопировать кодJAVASCRIPT

console.log('казнить' === 'помиловать'); *// false — это разные строки*

console.log('казнить' !== 'помиловать'); *// true*

console.log('2' !== 2); *// true — строка "2" не равна числу 2*

Операции сравнения всегда возвращают одно из двух значений: true или false.

## Условия

Простейшая условная конструкция запускает код, если условие, на которое она настроена, имеет значение true. Объявляется она ключевым словом if. Синтаксис конструкции if:

Скопировать кодJAVASCRIPT

if (условие) {

*// тело if*

}

Тело — это код, который запускается, если значение условия — true. Пример: условная конструкция выводит в консоль весёлый смайлик, если значение переменной merry равно true.

Скопировать кодJAVASCRIPT

let merry = true;

if (merry) {

console.log('😃');

}

*// 😃*

Код тела заключён в фигурные скобки. После них точки с запятой не нужны.

Если нужна реакция и на случай, когда условие ложно, то объявляют код с ключевым словом else. Вот скрипт, который выводит весёлый смайлик, когда merry — true. В остальных случаях выводится смайлик с нейтральным лицом.

Скопировать кодJAVASCRIPT

let merry = false;

if (merry) {

console.log('😃');

} else {

console.log('😐');

}

*// 😐*

Но мир не белый и чёрный, условий может быть больше. На случаи, не подходящие ни под if, ни под все остальные else, в JavaScript существует конструкция else if. Её синтаксис аналогичен правилам для if, а её место — между телом if и телом else:

Скопировать кодJAVASCRIPT

let teslaStockPrice = 644;

if (teslaStockPrice > 800) {

console.log('Акцию пора продавать');

} else if (teslaStockPrice > 650) {

console.log('Пока держим, пусть ещё подрастёт');

} else if (teslaStockPrice > 500) {

console.log('Как дёшево, надо ещё докупить');

} else {

console.log('Беру на все');

}

*// "Как дёшево, надо ещё докупить"*

*// PS: условная конструкция не является*

*// инвестиционной рекомендацией*

В многоэтажной конструкции самые важные условия должны помещаться выше, а главным остаётся первое, при ключевом слове if. Ведь верхние проверки выполняются первыми.

## Операции НЕ, И, ИЛИ

В 1847 году британский математик Джордж Буль стал решать логические задачи уравнениями, обозначив логическую единицу словом true (англ. «истинный»), а логический ноль словом false («ложный»). Такая алгебра в честь Буля называется булева алгебра, а особый тип данных, включающий значения true и false — булевы значения.

В примере со смайликами мы определили переменную merry и присваивали ей значение true или false, теперь мы знаем, что это называется булева переменная.

Скопировать кодJAVASCRIPT

let merry = true;

let christmas = false;

Эти переменные отличаются от строк или чисел, но и с ними можно выполнять операции.

**Логическое НЕ** — отрицание, делает из истины ложь, а из лжи — истину. Записывается эта операция с помощью восклицательного знака !:

Скопировать кодJAVASCRIPT

let merry = true;

console.log(!merry); *// false*

let christmas = false;

console.log(!christmas); *// true*

**Логическое И** вернёт истину только когда две булевы переменные истинны. Записывается эта операция с помощью двух амперсандов &&:

Скопировать кодJAVASCRIPT

let merry = true;

let christmas = false;

console.log(merry && christmas); *// false*

**Логическое ИЛИ** вернёт истину, когда хотя бы одно из булевых переменных истинно. Записывается эта операция с помощью двух прямых слешей ||:

Скопировать кодJAVASCRIPT

let merry = true;

let christmas = false;

console.log(merry || christmas); *// true*

Всё это позволяет записывать более сложные условия для инструкций if:

Скопировать кодJAVASCRIPT

let temperature = 36.6;

if (temperature > 36.3 && temperature <= 37.1) {

console.log('Ваша температура в пределах нормы');

} else {

console.log('Срочно обратитесь к врачу!');

}

Правильный расчёт високосных годов ведётся по правилу:

* год, номер которого кратен 400, — високосный;
* остальные годы, номер которых кратен 100, — невисокосные;
* остальные годы, номер которых кратен 4, — високосные.

Дополните условие if (...), чтобы проверка на «високосность» была настоящей. Используйте булевы операции И (&&) и ИЛИ (||).

let year = 2019;

if (year % 400 === 0 || year % 4 === 0 && year % 100 !== 0 ) {

console.log(year + ' год - високосный');

}

else {

console.log(year + ' год - невисокосный');

}

# Массивы, создание, доступ по индексу, длина массива

Массивы — структуры данных, которые могут хранить не одно, а несколько значений.

Массивы определяют квадратными скобами, а элементы перечисляют внутри скобок через запятую:

Скопировать кодJAVASCRIPT

let months = ['Январь', 'Февраль', 'Март', 'Апрель', 'Май', 'Июнь', 'Июль', 'Август', 'Сентябрь', 'Октябрь', 'Ноябрь', 'Декабрь'];

*// массив, в котором собраны все месяцы в году*

Номер элемента в массиве называют индексом.

Получить отдельный элемент можно по его индексу в квадратных скобках. Причём отсчёт ведётся с нуля: индекс первого элемента 0, а второго 1. Февраль, второй месяц в году, будет иметь индекс 1.

Скопировать кодJAVASCRIPT

months[1]; *// 'Февраль'*

У массивов есть свойство length, которое возвращает длину массива. Обратиться к свойству length можно двумя способами. Первый — по имени переменной. Для этого слово length пишут после названия переменной, в которой находится массив. Второй способ — написать слово length через точку прямо после квадратных скобок массива. Индекс последнего элемента всегда на единицу меньше длины массива.

Скопировать кодJAVASCRIPT

months.length; *// 12*

[].length; *// 0 - пустой массив не содержит элементов, его длина 0*

[1, 2, 3].length; *// 3 — в массиве три элемента, его длина 3*

В конце массива добавьте новый элемент: Помыть посуду.

Обращайтесь к элементам массива по индексу.

Выведите результат в консоль.

Запись элемента в массив происходит так:

morningList[номер\_элемента] = новое\_значение;

Если вместо номера элемента указать morningList.length, то новое значение запишется в конец массива.

# Циклы for, while, do...while

Цикл — это способ запустить один и тот же код несколько раз. Каждый повтор называется итерацией (от латинского itero — «опять»). Цикл состоит из условия и тела. Тело цикла — это код, который выполняется, пока выполняется условие цикла. И способов задать это условие несколько.

## Цикл с неизвестным числом повторений while

Первый тип цикла — цикл с неизвестным числом повторений. Этот цикл выполняется до тех пор, пока выполняется какое-то условие. Такой цикл объявляется словом while:

Скопировать кодJAVASCRIPT

while (условие) {

*// тело цикла*

}

Тело цикла — это те шаги, которые будут выполняться повторно. Тело цикла должно быть организовано так, чтобы условие рано или поздно перестало выполняться. Иначе цикл будет выполняться бесконечно.

Следующий цикл напечатает все чётные цифры от 10 до 20:

Скопировать кодJAVASCRIPT

let number = 10;

while (number <= 20) {

console.log(number);

number += 2;

}

Условие цикла содержит только проверку переменной number. Пока эта переменная не больше 20, цикл выполняется. Но тело цикла меняет значение number — прибавляет к нему 2 при каждой итерации. Поэтому после 6 повторений переменная number станет больше 20, и цикл прекратится.

## Цикл с известным числом повторений for

Второй тип цикла — цикл с известным числом повторений. Вот его синтаксис:

Скопировать кодJAVASCRIPT

for (переменная цикла и её исходное значение; условие выполнения цикла; изменение переменной после каждой итерации) {

*// тело цикла*

}

Выражение в скобках состоит из трёх частей. Сперва задаётся переменная, затем условие выполнения цикла, и в конце — как переменная должна меняться после каждого повторения.

Разберём подробнее, начиная со второй части — условия.

1. Условие выполнения цикла for — почти то же самое, что условие цикла while. Пока оно выполняется, выполняется и цикл. Но в цикле for условие всегда связано с особой переменной — счётчиком цикла.
2. Счётчик цикла — переменная, которая определяет число повторений. В начальной точке цикла мы задаём первоначальное значение этой переменной, а затем проверяем перед каждой итерацией, соответствует ли счётчик цикла условию. Если соответствует, цикл совершает ещё одно повторение. Если нет — не совершает.
3. Последняя часть внутри круглых скобок — шаг изменения переменной. Он определяет, на сколько счётчик цикла изменяется после каждого повторения.

Таким образом, сначала мы задаём исходное значение счётчика. Затем устанавливаем условие выполнения цикла. И затем описываем, как счётчик цикла меняется при каждой итерации.

Рассмотрим на примере:

Скопировать кодJAVASCRIPT

for (let i = 0; i <= 10; i = i + 1) {

console.log(i);

}

let i = 0 — создание переменной-счётчика;

i <= 10 — условие цикла: пока i меньше или равно 10, счётчик будет выполняться.

i = i + 1 — инкремент, он определяет, на сколько увеличивается переменная i после каждого повторения.

Цикл for —  самый популярный из всех циклов. Его вам предстоит использовать чаще всего.

## Цикл do...while

Третий тип цикла — do...while. Это такой же цикл как while, но с одним отличием: условие проверяется после каждой итерации, а не до неё:

Скопировать кодJAVASCRIPT

do {

<тело цикла>

} while (условие);

Это гарантирует, что тело цикла выполнится хотя бы один раз.

Пример цикла, условие которого ложно с самого начала:

Скопировать кодJAVASCRIPT

let number = 21;

do {

console.log(number);

} while (number <= 20);

Значение number вывелось в консоль, несмотря на то, что 21 не меньше 20. С циклом while такое бы не сработало.

**Функции: определение и вызов**

Функция — именованная часть кода. Она позволяет описать участок кода в одном месте и затем обращаться к нему из другого. Это структурирует код и делает его более читаемым. Но главное: функции позволяют использовать один и тот же код повторно.

Функция имеет имя и тело. Чтобы объявить функцию, пишут ключевое слово function, имя функции с круглыми скобками, а затем — её тело внутри фигурных скобок:

Скопировать кодJAVASCRIPT

function имяФункции() {

*// тело функции*

}

Опишем функцию, которая рисует в консоли котёнка:

Скопировать кодJAVASCRIPT

function consoleKitten() {

let a = ' Λ \_ Λ';

let b = ' (=චᆽච=)==∫';

let c = ' ˉ ˉ ˉ ˉ';

console.log(a);

console.log(b);

console.log(c);

}

Функция рисования котёнка готова. Но котёнка в консоли пока не оказалось. Дело в том, что пока мы только описали функцию, но не запустили её. Чтобы тело функции выполнилось, функцию нужно вызывать. Для этого пишут её имя и затем ставят круглые скобки:

Скопировать кодJAVASCRIPT

consoleKitten();

*// Λ \_ Λ*

*// (=චᆽච=)==∫*

*// ˉ ˉ ˉ ˉ*

**Параметры функции**

Параметры — переменные, которыми может пользоваться только сама функция.

В коде тела функции прописываются действия с ними, как с обычными переменными.

Объявим функцию keepScore, которая сообщает итог матча. Она принимает два параметра — ours и theirs. Это число голов, забитое «нашими» и соперником.

Скопировать кодJAVASCRIPT

function keepScore(ours, theirs) {

*// Проверим, забили наши больше голов или нет:*

if (ours > theirs) {

console.log('Выиграли! 😃 Счёт ' + ours + ':' + theirs);

*// Если наши забили не больше голов,*

*// то может столько же? Проверим:*

} else if (ours === theirs) {

console.log('Ничья. 😐 Счёт ' + ours + ':' + theirs);

*// Если два предыдущих условия не выполнены,*

*// стало быть, наши забили меньше голов :(*

} else {

console.log('Продули... 😢 Счёт ' + ours + ':' + theirs);

}

}

Результат матча можно указать прямо в коде вызова:

Скопировать кодJAVASCRIPT

keepScore(10, 8);

*// функция keepScore подставит вместо переменной ours значение 10,*

*// а вместо theirs — значение 8. Затем выполнит все проверки и выдаст результат:*

*// 'Выиграли! 😃 Счёт 10:8'*

При вызове функции можно передать ей как параметры значения переменных. Для этого параметры перечисляют внутри круглых скобок при вызове:

Скопировать кодJAVASCRIPT

let a = 10;

let b = 8;

keepScore(a, b);

**Возвращаемое значение**

Функции могут намного больше, чем просто выводить сообщения в консоль. Самое главное, что функция по своему алгоритму превращает исходные данные, такие как параметры или аргументы, в результаты.

Функциям обычно поручают преобразовывать данные, но не определять, где они применяются — в консоли, во всплывающем окне или где-то ещё. Преобразованные данные возвращают скрипту для дальнейшего использования.

Возвращаемые значения указывает оператор return:

Скопировать кодJAVASCRIPT

function sayHello(name) {

*// Преобразуем входные данные*

let greeting = 'Привет, ' + name;

*// Возвращаем результат*

return greeting;

}

let alisaGreeting = sayHello('Алиса');

console.log(alisaGreeting); *// "Привет, Алиса"*

В примере функция добавляет к строке 'Привет, ' параметр, который мы передаём при вызове ('Алиса'). Получившаяся строка должна стать результатом работы функции. Об этом сообщает оператор return.

Когда функция вызвана, мы получаем этот результат и решаем, что с ним делать. В примере мы сначала записали его в переменную, а затем вывели в консоль.

Приветствие может понадобиться снова. Не обязательно в консоли, например, в интерфейсе сайта. Мы легко сможем вызвать sayHello ещё раз.

**Как прервать работу функции**

Директива return означает выход из функции. В теле функции код ниже строки с оператором return не исполняется:

Скопировать кодJAVASCRIPT

function sayHello(name) {

let greeting = 'Привет, ' + name;

return greeting;

*// этот код не выполнится,*

*// так как он после return*

console.log(greeting);

}

А return без значения просто говорит функции «хватит»:

Скопировать кодJAVASCRIPT

function sayHello(name) {

if (name === '') {

return; *// если имя — пустая строка, выйдем из функции*

}

let greeting = 'Привет, ' + name;

return greeting;

}

Если передать такой функции пустую строку, она прекратит работу. Функция при этом вернёт специальное значение undefined:

Скопировать кодJAVASCRIPT

let emptyGreeting = sayHello('');

console.log(emptyGreeting); *// undefined*

Что такое undefined мы расскажем позже.

**Объекты**

Объекты — важнейшая концепция JavaScript. Они представляют собой структуру для хранения данных: чисел, строк, массивов, других объектов и даже функций. Объекты применяют повсеместно, потому что они позволяют организовать любой набор данных в иерархическую структуру. Мы будем ещё много говорить об объектах и часто ими пользоваться. Поэтому прочитайте этот урок очень внимательно.

Объекты состоят из пар «ключ — значение». Значение — это данные, которые мы хотим записать. Значением может быть строка, число, булево значение, массив, другой объект или функция. Ключ — это уникальное имя этого значения. Ключ играет ту же роль, что и имя переменной. По нему мы можем обратиться к значению.

Создадим объект myObject с четырьмя ключами:

Скопировать кодJAVASCRIPT

let myObject = {

stringKey: 'значение',

numberKey: 4,

booleanKey: true,

methodKey: function consoleKitten() {

console.log('kitten!');

}

};

Пары «ключ — значение» делятся на два типа: свойства и методы.

Если значение представляет собой функцию, такую пару называют методом.

Если значение — строка, число, булево значение, массив или объект, такую пару называют свойством.

Скопировать кодJAVASCRIPT

let myObject = {

stringKey: 'значение', *// это свойство*

numberKey: 4, *// это тоже свойство*

booleanKey: true, *// и это свойство*

methodKey: function consoleKitten() { *// а вот это метод*

console.log('kitten!');

}

};

Создав объект, мы наполняем его свойствами и методами. Затем пользуемся ими — вызываем методы и получаем доступ к свойствам. Чтобы получить доступ к свойству, его имя записывают через точку после имени объекта:

Скопировать кодJAVASCRIPT

myObject.stringKey;

Другой способ обратиться к свойству — указать имя свойства в кавычках и в квадратных скобках:

Скопировать кодJAVASCRIPT

myObject['numberKey'];

Если объект уподобить ящику с инструментами, то фигурные скобки — стенки ящика. После них нужна точка с запятой. Лежащие внутри ящика инструменты — пары «ключ-значение» — разделены запятыми. Это очень важно, без запятой ничего работать не будет.

# Введение в примитивы

В предыдущих темах вы уже сталкивались с примитивами — числами и строками. Пора погрузиться в эту тему глубже и узнать больше про другие примитивы и их особенности.

Иногда console.log при вызове может возвращать "undefined" и "null". В этой теме мы сорвём все покровы и расскажем про зоны ответственности каждого из этих типов данных. Понимание их значений поможет быстрее находить ошибки. И, не глядя в код, отвечать на вопросы будущих стажеров на вашей работе: «Какого чёрта этот метод вернул мне null?». Второе — важнейший навык :)

Кроме этого, вы узнаете больше о нескольких методах работы с числами и строками. К методам работы с примитивными типами данных мы будем обращаться ещё и в следующем спринте. Для программиста типы данных и работа с ними — своего рода таблица умножения — применять нужно не очень часто, но знать обязательно.

Но так ли они примитивны? Или они не те, за кого себя выдают?

**Примитивные типы данных. Оператор typeof**

В JavaScript данные делятся на:

* объекты, имеющие свойства пары: «ключ + значение»;
* примитивы (элементарные типы), у которых есть только значение.

Примитивных типов шесть:

* строки (тип “string”);
* числа (тип “number”);
* булевы значения true и false (тип “boolean”);
* undefined (тип “undefined”);
* null (тип “null”);
* символы (тип “symbol”).

В этой теме речь пойдёт обо всех примитивах, кроме символов.

**Определение типа данных. Оператор typeof**

Тип данных определяют оператором typeof. Оператор typeof возвращает строку:

Скопировать кодJAVASCRIPT

typeof 10; *// "number"*

typeof 'Hello World!'; *// "string"*

typeof true; *// "boolean"*

typeof undefined; *// "undefined"*

typeof работает без скобок, но их ставят, когда нужно определить тип целого выражения:

Скопировать кодJAVASCRIPT

typeof (10 + 5) *// "number"*

typeof 10 + 5 *// "number5"*

*/\* во втором случае сначала сработал typeof 10,*

*вернул строку "number", а оператор сложения произвёл*

*конкатенацию, и получилась строка "number5" \*/*

Работа typeof выглядит простой и понятной, но есть 3 случая, когда результат работы typeof неочевиден:

Скопировать кодJAVASCRIPT

typeof NaN; *// "number". Да, "Not a Number" имеет тип данных "number".*

typeof null; *// "object". Это даже было признано официальным багом JavaScript. Его решили не исправлять, чтобы не сломать уже написанный код.*

typeof function () {} *// "function". Хоть такого типа и нет.*

Помните об этих случаях, если решите использовать typeof в своём коде. Иначе можно спутать объект с null.

# Два одиноких типа данных. undefined и null

В JavaScript есть два типа данных, состоящих из единственного значения, — undefined и null.

## Значение не определено. undefined

Если в объявленной переменной ничего не хранится, JavaScript записывает в неё специальное значение undefined. undefined — отдельный тип данных, который состоит всего из одного значения:

Скопировать кодJAVASCRIPT

*/\* Объявим переменную meaningOfLife*

*Значение ей пока не присвоено. \*/*

let meaningOfLife;

*// Тип значения переменной meaningOfLife пока что undefined.*

typeof meaningOfLife; *// "undefined"*

Значение undefined можно задать и вручную:

Скопировать кодJAVASCRIPT

const meaningOfLife = undefined;

console.log(meaningOfLife); *// undefined*

Если явно не указать в функции, что она возвращает, она вернёт undefined:

Скопировать кодJAVASCRIPT

function searchForTheMeaningOfLife() {

const meaningOfLifeFound = true;

}

const meaningOfLife = searchForTheMeaningOfLife();

*/\* Поскольку функция ничего не возвращает,*

*в переменную meaningOfLife попадёт undefined. \*/*

console.log(meaningOfLife); *// undefined*

Если переменная равна undefined, значит она объявлена, но в неё ничего не записано. То же значение возвращается при обращении к несуществующему свойству объекта:

Скопировать кодJAVASCRIPT

const myMind = {};

console.log(myMind.location) *// undefined*

## Значения нет. null

null — это особое значение и отдельный тип данных. null сигнализирует, что значения нет.

Некоторые встроенные функции и методы возвращают null, когда не могут вернуть что-то осмысленное. Знакомый вам querySelector возвращает null, если не нашёл на странице элемент с искомым селектором:

Скопировать кодJAVASCRIPT

const element = document.querySelector('.non-existing-class');

if (element === null) {

console.log('Элемента с таким классом нет на странице');

}

Оператор typeof работает с null некорректно, говорит что это объект:

Скопировать кодJAVASCRIPT

console.log(typeof null); *// "object"*

Когда-то это было добавлено в спецификацию и реализовано в браузерах. Такое поведение признано ошибкой, но исправить её уже нельзя. Код сайтов, на которых это используется, может сломаться. Поэтому терпим.

# Строки

В этом уроке вы научитесь получать длину строки и её отдельные символы.

## Определение длины строки. Свойство length

У любого текста есть длина. Это значение свойства length, равное количеству символов в строке:

Скопировать кодJAVASCRIPT

console.log('АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ'.length); *// 33*

const greekAlphabet = 'αβγδεζηθικλμνξοπρστυφχψω';

console.log(greekAlphabet.length); *// 24*

## Получение символа строки по индексу

Квадратные скобки — простой способ получить символ из строки по его индексу, то есть обратиться к нему как к элементу массива. Индекс первого символа в строке — ноль:

Скопировать кодJAVASCRIPT

console.log('эспрессо'[0]); *// "э"*

*/\* можно таким же способом запросить символ*

*обращением к переменной, в которой сохранена строка: \*/*

const wrongWord = 'экспрессо';

console.log(wrongWord[1]); *// "к"*

*/\* Если запросить слишком большой индекс, получим undefined \*/*

console.log('экспрессо'[20]); *// undefined*

Это новые методы работы со строками, которые могут пригодиться в работе. Не забывайте о них.

Акростих — это такое стихотворение, в котором первые буквы каждой строки составляют слово или фразу. В 1917 году Николай Гумилёв посвятил Анне Ахматовой акростих, строчки которого мы сделали элементами массива.

Допишите код так, чтобы отобразить в консоли слова, зашифрованные в этом акростихе:

const acrostic = [

'Ангел лёг у края небосклона.',

'Наклонившись, удивлялся безднам.',

'Новый мир был синим и беззвездным.',

'Ад молчал, не слышалось ни стона.',

' ',

'Алой крови робкое биенье,',

'Хрупких рук испуг и содроганье.',

'Миру лав досталось в обладанье',

'Ангела святое отраженье.',

'Тесно в мире! Пусть живёт, мечтая',

'О любви, о грусти и о тени,',

'В сумраке предвечном открывая',

'Азбуку своих же откровений.'

];

*/\* Объявляем переменную, в которой*

*будет собрана итоговая строка \*/*

let annaAkhmatova = '';

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

*/\* Впишите код сюда \*/*

}

console.log(annaAkhmatova);

В каждой итерации цикла складывайте первый символ очередной строки со значением переменной:

annaAkhmatova += acrostic[i][0];

let annaAkhmatova = ''; // Объявляем переменную, в которой будет собрана итоговая строка

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

annaAkhmatova += acrostic[i][0];

}

console.log(annaAkhmatova);

# Числа и специальные числовые значения

Мы уже говорили о числах: учились, как их складывать, перемножать, возводить в степень. Пришло время понять, чем они являются для JavaScript и как работают.

В этом уроке расскажем о двух специальных значениях: бесконечностях и не числах.

### Бесконечность. Метод Number.isFinite

В JavaScript две бесконечности — Infinity и -Infinity. Это самое большое и самое малое числовые значения в языке:

Скопировать кодJAVASCRIPT

*// деление на 0 — это бесконечность*

25 / 0; *// Infinity*

*// если разделить на 0 отрицательное число,*

*// получим минус бесконечность*

-25 / 0; *// -Infinity*

*// бесконечность минус что угодно — бесконечность*

Infinity - 1000000000; *// Infinity*

Работа с бесконечностями непрозрачна, лучше ими не пользоваться:

Скопировать кодJAVASCRIPT

Infinity + -Infinity; *// NaN*

Infinity \* 0; *// NaN*

Infinity \* -1; *// -Infinity*

Infinity \* -Infinity; *// -Infinity*

**Метод Number.isFinite проверяет код на «бесконечность».** Если Number.isFinite передать на вход любую из бесконечностей, он вернёт false, конечное число — true:

Скопировать кодJAVASCRIPT

Number.isFinite(Infinity); *// false*

Number.isFinite(-Infinity); *// false*

Number.isFinite(1703); *// true*

**NaN — Not a Number. Метод**Number.isNaN

Если JavaScript не знает, как посчитать результат арифметического выражения, он сообщает об этом специальным значением — NaN.

Например, NaN возвращается, если умножить число на строку:

Скопировать кодJAVASCRIPT

console.log(10 \* 'десять'); *// NaN*

То же самое произойдёт и в других арифметических операциях со строкой или с undefined. Исключением является только сложение со строкой — оно приводит к конкатенации:

Скопировать кодJAVASCRIPT

console.log(10 / 'десять'); *// NaN*

console.log(10 - 'десять'); *// NaN*

console.log(10 + undefined); *// NaN*

*// а здесь произойдёт конкатенация*

console.log(10 + 'десять'); *// "10десять"*

NaN относится к числовому типу данных, пусть и называется «не число»:

Скопировать кодJAVASCRIPT

console.log(typeof NaN); *// "number"*

Забавное свойство NaN — оно не равно ничему, даже самому себе:

Скопировать кодJAVASCRIPT

console.log(NaN === NaN); *// false*

Из-за этого нельзя напрямую проверить, оказался ли результат вычисления не числом. Для решения этой проблемы создан метод Number.isNaN.

Когда методу Number.isNaN передают как параметр какое-нибудь выражение, он отвечает true или false на вопрос: «Правда ли, что этот параметр — NaN?»

Скопировать кодJAVASCRIPT

Number.isNaN(NaN); *// true*

Number.isNaN(0 / 0); *// true*

Из любого числа в JavaScript можно получить NaN и Infinity. Попробуйте сделать это и выведите результат в консоль:

const *initialNumber =* 451*;*

*// любая арифметическая операция (кроме сложения с числом или строкой)*

console.log(initialNumber \* 'хочу стать не-числом'); *// NaN*

*// деление на ноль*

console.log(initialNumber / 0); *// Infinity*