**JavaScript**

alert – вызывает уведомление на сайте.

“use strict” – включает в коде строгий режим.

(Записывается как библиотеки в си , первой строкой)

Т.е если переменную , не объявили с let или var, выдаст ошибку.

Переменные объявляются через let или var

let number = 0; var number = 0;

Типы данных:

1)  Undefined – не определенный тип, объявлена, но не присвоено никакого значения.(Не была определена)

2)  Null – тип, определяет собой ничего

3)  Boolean – true && false

4)  Number – числа. Не может содержать больше, чем

 9007199254740991

5) BigInt – число больше, чем Number. Записывается с окончанием “n” : 12312312312n

6) String – строка , есть три типа кавычек – “ ”, ‘ ’, ` `.

“” и ‘’ – одинаковы , а `` могут содержать в себе некие выражения:

let user = `Возраст ${userAge}`;

7)  Symbol – символы (‘A’);

Объявляется

let user = Symbol("1");

8)  Object – более, сложный тип данных .

Объявляется с помощью {}

Let user =

{

    name: “Dysa”, // Свойства . Ключ + Значение

    Age: 36

}

9)  Function – обычная функция, которая может быть присвоена к переменной.

Let funcVariable = function name (params) {

    //Code

}

Планирование функций:

    setTimeout(вызывает функцию один раз через определенный интервал)

    serInterval(вызывает функцию регулярно, повторяя через опредёленный интервал времени)

NaN – это вычислительная ошибку.

Alert (“Hi” / 2); // NaN

Infinity – математическая бесконечность.

Alert ( 1 / 0); // Infinity

Оператор typeof позволяет нам увидеть, какой тип данных сохранён в переменной

•   Имеет 2 формы: typeof x или typeof(x).

•   Возвращает строку с именем типа

•   Для null возвращается object

Функция prompt  - модальное окно с текстом, полем ввода для текста и кнопками ОК/Отмена.

Let age = prompt(‘How old are u’,100);

Вызов prompt возвращает текст , или null, если ввод отменён.

Функция confirm  - отображает модальное окно с текстом вопроса и двумя кнопками  ОК и Отмена.

Let age = confirm(‘Ты здесь главный?’);

Вызов confirm возвращает true or false.

Базовые Операторы

Возведение в степень 2\*\*2.

Сложение строк:

let s = “my” + “str”

s = mystr;

    alert(‘1’ + 2) // 12

    alert (2 + 2 + ‘1’) // 221

Унарный + преобразует не число в число.

     Alert (+true); // 1

    Alert(+””);  // 0

Постфиксная и Префикная форма Инкремента и Дискремента(++ и --).

    let count = 1;

    alert (2 \* ++count) // 4

    alert (2 \* count++) // 2

Операторы сравнения

alert (null === undefiend) // false // спец правило языка

alert (null == undefiend) // true

Условие .?.

Тот же самый if, только запись другая

    let result = (age > 18) ? true: false.

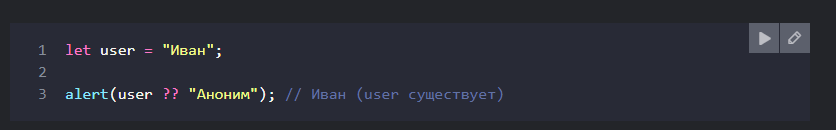
Логические операторы

alert (!!null); // false

alert(!!(“Dysa”)); // true

Оператор нулевого слияния ??

Оператор ?? нужен для того , чтобы задать значение по умолчанию для потенциально неопределённой переменной.



ОБЬЕКТЫ

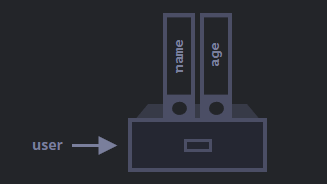
Обьект – это тип данных хранящий в себе коллекцию различных значений.

let user =

{

Name: “Josh”,

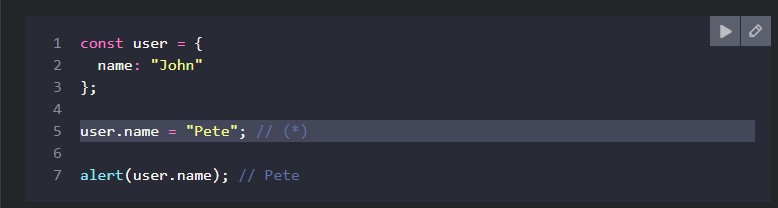
Age: 21



};

Чисто как столик с папками, а папки уже хранят значение.

Похожа на листы из си



Проход в объектах: работает и с квадратными скобками:

let user = {};

user["likes birds"] = true;

alert(user["like birds"]);

delete user["like birds"];

Проход в объектах:

let user =

{

name:"Josh",

age:30,

isAdmin: true

};

for (let key in user)// проход

{

alert(key)

alert(1);

alert(user[key]);

}

Объединение объектов

let user = { name: "John" };

let permissions1 = { canView: true };

let permissions2 = { canEdit: true };

Object.assign(user, permissions1, permissions2);

Объекты присваиваются и копируются по ссылке.

Другими словами, переменная хранит не «значение объекта», «ссылку» (адрес в памяти) на это значение Таким образом, копирование такой переменной

или передача её в качестве аргумента функции копирует эту ссылку, а не сам объект.

Все операции с использованием скопированных ссылок (например, добавление/удаление свойств) выполняются с одним и тем же объектом.

Чтобы создать «реальную копию» (клон), мы можем использовать Object.assign для так называемой «поверхностной копии» (вложенные объекты копируются по ссылке) или

функцию «глубокого клонирования», такую как \_.cloneDeep(obj).

Сборка мусора

Сборка мусора выполняется автоматически.

Мы не можем ускорить или предотвратить её.

Объекты сохраняются в памяти, пока они достижимы.

Если на объект есть ссылка – вовсе не факт, что он является достижимым (из корня):

набор взаимосвязанных объектов может стать недоступен в целом.

Ключевое слово «this» в методах

Значение this – это объект «перед точкой», который используется для вызова метода.

Задача №1

let calculator = {

    read()

    {

        this.number = +prompt("Введите первое значение",0);

this.number1 = +prompt("Введите второе значение", 0);

    },

    sum()

    {

        return  this.number +   this.number1;

    },

    mul()

    {

        return  this.number \*   this.number1;

    }

};

calculator.read();

alert(calculator.sum());

alert(calculator.mul());

Задача №2

let ladder =

{

step : 0,up()

{

this.step++;

return this;

},

down()

{

this.step--;

return this;

},

showStep: function()

{

alert(this.step);

return this;

}

};

ladder.up().up().down().showStep().down().showStep();

Задача №3

let userInfo ={

    name: "Vasya",

    age: 30,

    address:{

    city: "Uzhorod",

    street: "Freedom"

    },

    showInfo()

    {

console.log(`${this.name}, ${this.age} лет. Адрес: г.${this.address.city},${this.address.street}`)

    }

}

userInfo.showInfo();

Конструктор, оператор "new"

Функция конструктор – имя конструкции должна начинаться с большой буквы

function UserInfo(name)

{

    this.name = name;

    this.age = 30;

}

console.log(new UserInfo('Вася'));

console.log(new UserInfo('Лена'));

function User(name)

{

    this.name = name;

    this.sayHi = function()

    {

        alert("My name: " + this.name);

    };

}

let josh = new User("Josh");

josh.sayHi();

function Accumulator(startingValue)

{

    this.value = startingValue;

    this.read = function()

    {

        this.value += +prompt("Введите значение:",0);

    };

}

let accumulator = new Accumulator(1); // начальное значение 1

accumulator.read(); // прибавляет введённое пользователем значение к текущему значению

accumulator.read();

alert(accumulator.value);

Опциональная цепочка '?.'

Синтаксис опциональной цепочки ?. имеет три формы:

1. Цепочка ?. позволяет безопасно получать доступ к вложенным свойствам.
2. Тем не менее, мы должны использовать ?. осторожно, только там,

где по логике кода допустимо, что левая часть не существует.

1. Чтобы он не скрывал от нас ошибки программирования, если они возникнут.

let key = "firstName";

let user1 = {

  firstName: "John"

};

let user2 = null;

alert( user1?.[key] ); // John

alert( user2?.[key] ); // undefined

Тип данных Symbol

Символ (symbol) – примитивный тип данных, использующийся для создания уникальных идентификаторов.

Символы создаются вызовом функции Symbol(), в которую можно передать описание (имя) символа.

Даже если символы имеют одно и то же имя, это – разные символы.

Если мы хотим, чтобы одноимённые символы были равны,

то следует использовать глобальный реестр:

вызов Symbol.for(key) возвращает (или создаёт) глобальный символ с key в качестве имени.

Многократные вызовы команды Symbol.for с одним и тем же аргументом возвращают один и тот же символ.

// Создаём символ id с описанием (именем) "id"

let id = Symbol("id");

мы можем обратиться к свойству symbol.description, чтобы вывести только описание:

let id = Symbol("id");

alert(id.description)

let user = {

    name:"Вася"

}

let id = Symbol("id");

user[id] = 1;

alert(id.description);

Object.assign, в отличие от цикла for..in, копирует и строковые, и символьные свойства:

let id = Symbol("id");

let user = {

  [id]: 123

};

let clone = Object.assign({}, user);

alert(clone[id] ); // 123

Глобальные символы

Итак, как мы видели, обычно все символы уникальны,

даже если их имена совпадают. Но иногда мы наоборот хотим,

чтобы символы с одинаковыми именами были одной сущностью.

Например, разные части нашего приложения хотят получить

доступ к символу "id", подразумевая именно одно и то же свойство.

// читаем символ из глобального реестра и записываем его в переменную

let id = Symbol.for("id"); // если символа не существует, он будет создан

// читаем его снова и записываем в другую переменную (возможно, из другого места кода)

let idAgain = Symbol.for("id");

// проверяем -- это один и тот же символ

alert( id === idAgain ); // true

Symbol.keyFor

Для глобальных символов, кроме Symbol.for(key),

который ищет символ по имени, существует обратный метод:

Symbol.keyFor(sym), который, наоборот, принимает глобальный символ и возвращает его имя.

К примеру:

// получаем символ по имени

let sym = Symbol.for("name");

let sym2 = Symbol.for("id");

// получаем имя по символу

alert( Symbol.keyFor(sym) ); // name

alert( Symbol.keyFor(sym2) ); // id

Преобразование объектов в примитивы

Symbol.toPrimitive

  let user = {

    name:"Josh",

    money : 1000,

    [Symbol.toPrimitive](hint) {

        alert(`hint: ${hint}`);

        return hint == "string" ? `{name: "${this.name}"}` : this.money;

    }

  };

  alert(user);

  alert(+user);

  alert(user + 500);

Преобразование объекта в примитив вызывается автоматически многими

встроенными функциями и операторами, которые ожидают примитив в качестве значения.

Существует всего 3 типа (хинта) для этого:

"string" (для alert и других операций, которым нужна строка)

"number" (для математических операций)

"default" (для некоторых других операторов, обычно объекты реализуют его как "number")

Спецификация явно описывает для каждого оператора, какой ему следует использовать хинт.

Алгоритм преобразования таков:

Сначала вызывается метод obj[Symbol.toPrimitive](hint), если он существует,

В случае, если хинт равен "string"

происходит попытка вызвать obj.toString() и obj.valueOf(), смотря что есть.

В случае, если хинт равен "number" или "default"

происходит попытка вызвать obj.valueOf() и obj.toString(), смотря что есть.

Все эти методы должны возвращать примитив (если определены).

На практике часто бывает достаточно реализовать только obj.toString() в качестве универсального метода для преобразований к строке, который должен возвращать удобочитаемое представление объекта для целей логирования или отладки.

Методы примитивов

toFixed(n) округляет число до n знаков после запятой.

let num = 1.23456;

alert(num.toFixed(2)); //1.23

str.toUpperCase(), который возвращает строку в верхнем регистре.

let str = 'qwerty';

alert(str.toUpperCase());// QWERTY

В JavaScript можно использовать букву "e",

чтобы укоротить запись числа. Она добавляется к числу и заменяет указанное количество нулей:

let billion = 1e9; // 1 миллиард, буквально: 1 и 9 нулей

alert(7.3e9 );  // 7.3 миллиардов (7,300,000,000)

let ms = 1e-6; // шесть нулей, слева от 1

Метод num.toString(base) возвращает строковое представление числа num в системе счисления base.

let num = 255;

alert( num.toString(16) );  // ff

alert( num.toString(2) );   // 11111111

alert( 123456..toString(36) );

Округление

Math.floor - Округление в меньшую сторону: 3.1 становится 3, а -1.1 — -2.

Math.ceil - Округление в большую сторону: 3.1 становится 4, а -1.1 — -1.

Math.round - Округление до ближайшего целого: 3.1 становится 3, 3.6 — 4, а -1.1 — -1.

Math.trunc -Производит удаление дробной части без округления: 3.1 становится 3, а -1.1 — -1.

Метод toFixed(n) округляет число до n знаков после запятой и возвращает строковое представление результата.

let num = 12.34;

alert( num.toFixed(1) ); // "12.3"

Обратите внимание, что результатом toFixed является строка.

Если десятичная часть короче, чем необходима, будут добавлены нули в конец строки:

let num = 12.34;

alert( num.toFixed(5) ); // "12.34000", добавлены нули, чтобы получить 5 знаков после запятой

Проверка: isFinite и isNaN

Infinity (и -Infinity) — особенное численное значение, которое ведёт себя в точности как математическая бесконечность ∞.

Эти числовые значения принадлежат типу number, но они не являются «обычными» числами, поэтому есть функции для их проверки:

isNaN(value) преобразует значение в число и проверяет является ли оно NaN:

alert( isNaN(NaN) ); // true

alert( isNaN("str") ); // true

Нужна ли нам эта функция? Разве не можем ли мы просто сравнить === NaN?

К сожалению, нет. Значение NaN уникально тем, что оно не является равным ни чему другому, даже самому себе:

alert( NaN === NaN ); // false

isFinite(value) преобразует аргумент в число и возвращает true, если оно является обычным числом, т.е. не NaN/Infinity/-Infinity:

alert( isFinite("15") ); // true

alert( isFinite("str") ); // false, потому что специальное значение: NaN

alert( isFinite(Infinity) ); // false, потому что специальное значение: Infinity

Иногда isFinite используется для проверки, содержится ли в строке число:

let num = +prompt("Enter a number", '');

// вернёт true всегда, кроме ситуаций, когда аргумент - Infinity/-Infinity или не число

alert( isFinite(num) );

Методы Number.isNaN и Number.isFinite – это более «строгие» версии функций isNaN и isFinite.

Они не преобразуют аргумент в число, а наоборот – первым делом проверяют,

является ли аргумент числом (принадлежит ли он к типу number).

Number.isNaN(value) возвращает true только в том случае,

если аргумент принадлежит к типу number и является NaN. Во всех остальных случаях возвращает false.

Number.isFinite(value) возвращает true только в том случае,

если аргумент принадлежит к типу number и не является NaN/Infinity/-Infinity. Во всех остальных случаях возвращает false.

Есть parseInt и parseFloat.

Они «читают» число из строки. Если в процессе чтения возникает ошибка, они возвращают полученное до ошибки число. Функция parseInt возвращает целое число, а parseFloat возвращает число с плавающей точкой:

alert( parseInt('100px') ); // 100

alert( parseFloat('12.5em') ); // 12.5

alert( parseInt('12.3') ); // 12, вернётся только целая часть

alert( parseFloat('12.3.4') ); // 12.3, произойдёт остановка чтения на второй точке

Math.random() - Возвращает псевдослучайное число в диапазоне от 0 (включительно) до 1 (но не включая 1)

alert( Math.random() ); // 0.1234567894322

alert( Math.random() ); // 0.5435252343232

alert( Math.random() ); // ... (любое количество псевдослучайных чисел)

Math.max(a, b, c...) / Math.min(a, b, c...) - Возвращает наибольшее/наименьшее число из перечисленных аргументов.

alert( Math.max(3, 5, -10, 0, 1) ); // 5

alert( Math.min(1, 2) ); // 1

Math.pow(n, power) - Возвращает число n, возведённое в степень power

alert( Math.pow(2, 10) ); // 2 в степени 10 = 1024

Строки

Свойство length содержит длину строки:

alert( `My\n`.length ); // 3

Доступ к символам

Получить символ, который занимает позицию pos, можно с помощью квадратных скобок: [pos]. Также можно использовать метод charAt: str.charAt(pos). Первый символ занимает нулевую позицию:

let str = `Hello`;

// получаем первый символ

alert( str[0] ); // H

alert( str.charAt(0) ); // H

// получаем последний символ

alert( str[str.length - 1] ); // o

Также можно перебрать строку посимвольно, используя for..of:

for (let char of "Hello") alert(char); // H,e,l,l,o (char — сначала "H", потом "e", потом "l" и т. д.)

Можно создать новую строку и записать её в ту же самую переменную вместо старой.

Например:

let str = 'Hi';

str = 'h' + str[1]; // заменяем строку

alert( str ); // hi

Методы toLowerCase() и toUpperCase() меняют регистр символов:

alert( 'Interface'.toUpperCase() ); // INTERFACE

alert( 'Interface'.toLowerCase() ); // interface

Если мы захотим перевести в нижний регистр какой-то конкретный символ:

alert( 'Interface'[0].toLowerCase() ); // 'i'

str.indexOf(substr, pos)- он ищет подстроку substr в строке str, начиная с позиции pos, и возвращает позицию, на которой располагается совпадение, либо -1 при отсутствии совпадений.

Например:

let str = 'Widget with id';

alert( str.indexOf('Widget') ); // 0, потому что подстрока 'Widget' найдена в начале

alert( str.indexOf('widget') ); // -1, совпадений нет, поиск чувствителен к регистру

alert( str.indexOf("id") ); // 1, подстрока "id" найдена на позиции 1 (..idget with id)

str.lastIndexOf(substr, position) - метод str.lastIndexOf(substr, position), который ищет с конца строки к её началу.

Он используется тогда, когда нужно получить самое последнее вхождение: перед концом строки или начинающееся до (включительно) определённой позиции.

includes, startsWith, endsWith

Более современный метод str.includes(substr, pos) возвращает true, если в строке str есть подстрока substr, либо false, если нет.

Это — правильный выбор, если нам необходимо проверить, есть ли совпадение, но позиция не нужна:

alert( "Widget with id".includes("Widget") ); // true

alert( "Hello".includes("Bye") ); // false

Необязательный второй аргумент str.includes позволяет начать поиск с определённой позиции:

alert( "Midget".includes("id") ); // true

alert( "Midget".includes("id", 3) ); // false, поиск начат с позиции 3

Методы str.startsWith и str.endsWith проверяют, соответственно, начинается ли и заканчивается ли строка определённой строкой:

alert( "Widget".startsWith("Wid") ); // true, "Wid" — начало "Widget"

alert( "Widget".endsWith("get") ); // true, "get" — окончание "Widget"

Получение подстроки

slice(start, end) - от start до end (не включая end)  можно передавать отрицательные значения

let str = "stringify"; // 'strin', символы от 0 до 5 (не включая 5)

alert( str.slice(0, 5) ); // 's', от 0 до 1, не включая 1, т. е. только один символ на позиции 0

alert( str.slice(0, 1) )

substring(start, end)   между start и end   отрицательные значения равнозначны 0

let str = "stringify";

// для substring эти два примера — одинаковы

alert( str.substring(2, 6) ); // "ring"

alert( str.substring(6, 2) ); // "ring"

// …но не для slice:

alert( str.slice(2, 6) ); // "ring" (то же самое)

alert( str.slice(6, 2) ); // "" (пустая строка)

substr(start, length)   length символов, начиная от start значение start может быть отрицательным

let str = "stringify";

// ring, получаем 4 символа, начиная с позиции 2

alert( str.substr(2, 4) );

Значение первого аргумента может быть отрицательным, тогда позиция определяется с конца:

let str = "stringify";

// gi, получаем 2 символа, начиная с позиции 4 с конца строки

alert( str.substr(-4, 2) );

Сравнение строк

1. Строчные буквы больше заглавных:

alert( 'a' > 'Z' ); // true

2. Буквы, имеющие диакритические знаки, идут «не по порядку»:

alert( 'Österreich' > 'Zealand' ); // true

Строки кодируются в UTF-16. Таким образом, у любого символа есть соответствующий код.

Есть специальные методы, позволяющие получить символ по его коду и наоборот.

str.codePointAt(pos) - возвращает код для символа, находящегося на позиции pos:

alert( "z".codePointAt(0) ); // 122

alert( "Z".codePointAt(0) ); // 90

String.fromCodePoint(code) - создаёт символ по его коду code

alert( String.fromCodePoint(90) ); // Z

Вызов str.localeCompare(str2) возвращает число, которое показывает, какая строка больше в соответствии с правилами языка:

alert( 'Österreich'.localeCompare('Zealand') ); // -1

Строки также имеют ещё кое-какие полезные методы:

str.trim() — убирает пробелы в начале и конце строки.

str.repeat(n) — повторяет строку n раз.

Массивы

Общее число элементов массива содержится в его свойстве length:

let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits.length ); // 3

В массиве могут храниться элементы любого типа.

Например:

// разные типы значений

let arr = [ 'Яблоко', { name: 'Джон' }, true, function() { alert('привет'); } ];

// получить элемент с индексом 1 (объект) и затем показать его свойство

alert( arr[1].name ); // Джон

// получить элемент с индексом 3 (функция) и выполнить её

arr[3](); // привет

Висячая запятая

Список элементов массива, как и список свойств объекта, может оканчиваться запятой:

let fruits = [

"Яблоко",

"Апельсин",

"Слива",

];

«Висячая запятая» упрощает процесс добавления/удаления элементов, так как все строки становятся идентичными.

Получение последних элементов при помощи «at»

let fruits = ["Apple", "Orange", "Plum"];

// то же самое, что и fruits[fruits.length-1]

alert( fruits.at(-1) ); // Plum

Методы pop/push, shift/unshift

Очередь – один из самых распространённых вариантов применения массива.

В области компьютерных наук так называется упорядоченная коллекция элементов, поддерживающая два вида операций:

push  - добавляет элемент в конец.

shift - удаляет элемент в начале, сдвигая очередь, так что второй элемент становится первым.

let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.shift() ); // удаляем Яблоко и выводим его

alert( fruits ); // Апельсин, Груша

unshift - добавляет элемент в начало массива:

let fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

alert( fruits ); // Яблоко, Апельсин, Груша

Методы push и unshift могут добавлять сразу несколько элементов:

let fruits = ["Яблоко"];

fruits.push("Апельсин", "Груша");

fruits.unshift("Ананас", "Лимон");

// ["Ананас", "Лимон", "Яблоко", "Апельсин", "Груша"]

alert( fruits );

Массивы поддерживают обе операции.

На практике необходимость в этом возникает очень часто.

Например, очередь сообщений, которые надо показать на экране.

Существует и другой вариант применения для массивов – структура данных, называемая стек.

Она поддерживает два вида операций:

push - добавляет элемент в конец.

let fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша

pop - удаляет последний элемент.

let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.pop() ); // удаляем "Груша" и выводим его

alert( fruits ); // Яблоко, Апельсин

Следует помнить, что в JavaScript существует 8 основных типов данных. Массив является объектом и, следовательно, ведёт себя как объект.

Одним из самых старых способов перебора элементов массива является цикл for по цифровым индексам:

        let arr = ["Яблоко", "Апельсин", "Груша"];

        for (let i = 0; i < arr.length; i++) alert( arr[i] );

Но для массивов возможен и другой вариант цикла, for..of:

        let fruits = ["Яблоко", "Апельсин", "Слива"];

        // проходит по значениям

        for (let fruit of fruits) {

        alert( fruit );

        }

Таким образом, самый простой способ очистить массив – это arr.length = 0;

Массивы по-своему реализуют метод toString, который возвращает список элементов, разделённых запятыми.

Например:

let arr = [1, 2, 3];

alert( arr ); // 1,2,3

alert( String(arr) === '1,2,3' ); // true

Давайте теперь попробуем следующее:

alert( [] + 1 ); // "1"

alert( [1] + 1 ); // "11"

alert( [1,2] + 1 ); // "1,21"

Методы массивов

arr.splice(index[, deleteCount, elem1, ..., elemN]) - Он начинает с позиции index, удаляет deleteCount элементов и вставляет elem1, ..., elemN на их место. Возвращает массив из удалённых элементов.

let arr = ["Я", "изучаю", "JavaScript"];

arr.splice(1, 1); // начиная с позиции 1, удалить 1 элемент

alert( arr ); // осталось ["Я", "JavaScript"]

let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];

// удалить 3 первых элемента и заменить их другими

arr.splice(0, 3, "Давай", "танцевать");

alert( arr ) // теперь ["Давай", "танцевать", "прямо", "сейчас"]

Метод splice также может вставлять элементы без удаления, для этого достаточно установить deleteCount в 0:

let arr = ["Я", "изучаю", "JavaScript"];

// с позиции 2

// удалить 0 элементов

// вставить "сложный", "язык"

arr.splice(2, 0, "сложный", "язык");

alert( arr ); // "Я", "изучаю", "сложный", "язык", "JavaScript"**}**

**Перебираемые объекты**

**Чтобы перебрать объект нужен метод**

[Symbol.iterator] и цикл for…of

let range = {

    from:1,

    to:5,

    [Symbol.iterator](){

        this.current = this.from;

        return this;

    },

    next()  {

        if(this.current <= this.to){

            return{done:false,value:this.current++}  }

        else{

            return{done:true};

        }}};

for(let num of range){

    alert(num);

}

1. Когда цикл for..of запускается, он вызывает этот метод один раз (или выдаёт ошибку, если метод не найден). Этот метод должен вернуть итератор – объект с методом next.
2. Дальше for..of работает только с этим возвращённым объектом.
3. Когда for..of хочет получить следующее значение, он вызывает метод next() этого объекта.
4. Результат вызова next() должен иметь вид {done: Boolean, value: any}, где done=true означает, что цикл завершён, в противном случае value содержит очередное значение.

**Строка - Перебираемые объекты**

**Проход и вывод всех элементов в строке**

for (let char of "test") alert( char ); // t, затем e, затем s, затем t

let str = '𝒳😂';

for (let char of str)  alert( char ); // 𝒳, а затем 😂

Как работает for…of простым языком в виде цикла while:

let str = "Hello";

let iterator = str[Symbol.iterator]();

while(true)

{

    let result = iterator.next();

    if (result.done) break;

    alert(result.value);

}

А вот объект, который является псевдомассивом, но его нельзя итерировать:

let arrayLike = {

    1:"Hello",

    2:",World",

    length:2};

for (let item  of arrayLike){}

Преобразовать итерируемые объекты и псевдомассивы можно методом Array.from

Есть универсальный метод Array.from, который принимает итерируемый объект или псевдомассив

и делает из него настоящий массив. После этого мы уже можем использовать методы массивов

Array.from(obj[, mapFn, thisArg])

Необязательный второй аргумент может быть функцией, которая будет применена к каждому элементу

перед добавлением в массив, а thisArg позволяет установить this для этой функции.

let array = Array.from(arrayLike, num => num \* num);

alert(arr); // 1,4,9,16,25

Map и Set

Map – это коллекция ключ/значение, как и Object. Но основное отличие в том, что Map позволяет использовать ключи любого типа.

Методы и свойства:

new Map() – создаёт коллекцию.

map.set(key, value) – записывает по ключу key значение value.

map.get(key) – возвращает значение по ключу или undefined, если ключ key отсутствует.

map.has(key) – возвращает true, если ключ key присутствует в коллекции, иначе false.

map.delete(key) – удаляет элемент (пару «ключ/значение») по ключу key.

map.clear() – очищает коллекцию от всех элементов.

map.size – возвращает текущее количество элементов.

Например:

let map = new Map();

map.set("1", "str1");    // строка в качестве ключа

map.set(1, "num1");      // цифра как ключ

map.set(true, "bool1");  // булево значение как ключ

// помните, обычный объект Object приводит ключи к строкам?

// Map сохраняет тип ключей, так что в этом случае сохранится 2 разных значения:

alert(map.get(1)); // "num1"

alert(map.get("1")); // "str1"

alert(map.size); // 3

Использование объектов в качестве ключей – одна из наиболее заметных и важных функций Map.

Это то что невозможно для Object. Строка в качестве ключа в Object – это нормально, но мы не можем использовать другой Object в качестве ключа в Object.

Перебор Map

Для перебора коллекции Map есть 3 метода:

map.keys() – возвращает итерируемый объект по ключам,

map.values() – возвращает итерируемый объект по значениям,

map.entries() – возвращает итерируемый объект по парам вида [ключ, значение], этот вариант используется по умолчанию в for..of.

let recipeMap = new Map([

    ["Огурец",500],

    ["Помидор",350],

    ["Лук",50],

]);

for(let vegetable of recipeMap.keys()) alert(vegetable);

for(let amount of recipeMap.values()) alert(amount);

for(let entry of recipeMap) alert(entry);

Если у нас уже есть обычный объект, и мы хотели бы создать Map из него, то поможет встроенный метод Object.entries(obj)

let obj = {

  name: "John",

  age: 30

};

let map = new Map(Object.entries(obj));

alert( map.get('name') ); // John

Мы можем использовать Object.fromEntries(obj), чтобы получить обычный объект из Map.

let map = new Map();

map.set('banana',1);

let obj = Object.fromEntries(map.entries());

alert(obj.orange);

Set

Объект Set – это особый вид коллекции: «множество» значений (без ключей), где каждое значение может появляться только один раз.

Его основные методы это:

new Set(iterable) – создаёт Set, и если в качестве аргумента был предоставлен итерируемый объект (обычно это массив), то копирует его значения в новый Set.

set.add(value) – добавляет значение (если оно уже есть, то ничего не делает), возвращает тот же объект set.

set.delete(value) – удаляет значение, возвращает true, если value было в множестве на момент вызова, иначе false.

set.has(value) – возвращает true, если значение присутствует в множестве, иначе false.

set.clear() – удаляет все имеющиеся значения.

set.size – возвращает количество элементов в множестве.

Основная «изюминка» – это то, что при повторных вызовах set.add() с одним и тем же значением ничего не происходит, за счёт этого как раз и получается, что каждое значение появляется один раз.

Например, мы ожидаем посетителей, и нам необходимо составить их список. Но повторные визиты не должны приводить к дубликатам. Каждый посетитель должен появиться в списке только один раз.

Множество Set – как раз то, что нужно для этого:

**serInterval(**вызывает функцию регулярно