**Примитивный тип данных**

является экземпляром определенного типа данных, таких как строковый, числовой, логический, null и undefined.

var one = 1;(создается до ее обьявления и действует глобально); let second = 2; ( используется один раз за счет чего нам не нужно хранить все переменные в данных ) const pi = 3.14;

var b = 1.234E+2; // вещественное число, эквивалентно 1.234 Х 10²

var x=1; //переменная целого типа

var y=1.5; //переменная вещественного типа

var z="abc"; //переменная типа строка, ковычка может быть " или ' разницы нет

var sym = Symbol(); новый тип данных

 Кавычка того же типа отключается с помощью символа обратного слэша \ (так называемая **escape-последовательность**):

var k=true; //переменная логического типа

**Нулевой тип** — данный тип имеет одно значение null, которое используется для представления несуществующих объектов.

**Неопределенный тип** — тип переменной underfined означает отсутствие первоначального значения переменной(но переменная создана), а также несуществующее свойство объекта.

Cпециальные числовые значения:

Infinity ( 4/0); NaN(“blow”/3)

**Сложный,Составные, комплексные типы данных, Обьекты**

состоят из более чем одного значения. К ним относятся объекты и особые типы объектов — массивы и функции.

**Объекты** содержат свойства(ключ , строки, свойства) и значение, елементы let person = { name: "John",age: 25,isMarried: false}

let person = { "name": "John","age": 25,"isMarried": false}

console.log(person["name"]);

person.name; - точечная нотация

**Существуют особые специальные типы обьектов** массивы и функции, error, date , регулярные выражения и **Обычные обьекты**

Обьекты не отсортированы по порядку как в числовых индексах (массивах)

Object.keys(object);

For (let key in person) {console.log(“свойство ” + key “имеет значение” + options[key]};

delete person.name;

**Object Date**

const now = new Date();

console.log(now.getFullYear());

console.log(now.getMonth());

console.log(now.getTimezoneOffset());

console.log(now.getUTCHours());

console.log(now.getTime());

console.log(now.setHours(18));

console.log(now);

**Определение время работы цикла**

const start = new Date();

for( let i = 0; i< 1000000; i++){

    let some = i \*\* 3;

}

const end = new Date();

alert(`Цикл отработал за ${end - start} миллисекунд! `);

**Массивы** представляют собой индексированный набор элементов

let arr = ['plum.png', 'orange.jpg'];

let arr [ 'plum.png', 'orange.jpg', [3,”5”], 4, “orange.png”]; - вложенный массив

arr[2][0]; равен 3

**Методы перебора массива :**

arr.forEach – function(значение, индекс, массив)

for (let key in array) **– ключ , значение** ,for(let key of array) - **значение**

arr.map

var numbers = [1, 4, 9];

var doubles = numbers.map(function(num) {

return num \* 2;

});

// теперь doubles равен [2, 8, 18], а numbers всё ещё равен [1, 4, 9]

arr.every/some

arr.filter

arr.reduse

Метод **call()** вызывает функцию с указанным значением this и индивидуально предоставленными аргументами.

**Псевдомассив**

**Функции** состоят из коллекции инструкций.

**Function declaration** – создается до начала выполнения скрипта , можно вызвать перед объявлением

function foo(){code}-

**Function expression** - создается когда попадется в поле видимости и вызывается только после обьявления

let foo = function(){code}

**Стрелочная функция**

**() =>**

**Если один параметр и можно поместить в одну строчку**

const double = a =>a \* 2;

console.log(double(4));

**Нет контекста вызова , берется контекст вызова родителя**

**Callback функция**

Вызывается в тот момент когда нам нужно, как параметр другой функции

function learnJs(lang, callback) {

    console.log( "Я учу "+ lang);

    callback();

}

function done() {

    console.log("Я прошел 3й урок!");

}

learnJs("JavaScript", done);

**Немедленное обращение к анонимной функции (**немедленно вызываемая функция)

<!-- My Script -->

<script>

function initialize(){

// foo is safely hidden within initialize, but...

var foo = '';

}

initialize();

</script>

<script>

// Using an anonymous function, and then immediately

// invoking it, hides my foo variable and guarantees

// no one else can call it a second time.

(function(){

var foo = '';

}()) // <--- the parentheses invokes the function immediately

</script>

CALL , APPLY

var user = {

firstName: "Василий",

surname: "Петров",

patronym: "Иванович"

};

function showFullName(firstPart, lastPart) {

alert( this[firstPart] + " " + this[lastPart] );

}

// f.call(контекст, аргумент1, аргумент2, ...)

showFullName.call(user, 'firstName', 'surname') // "Василий Петров"

showFullName.call(user, 'firstName', 'patronym') // "Василий Иванович"

**Ключевые слова(зарезервированные слова):**

confirm(“are you here?”);

prompt(, ,);

+prompt(, , ); + - преобразует из типа данных string в int

typeof();

строгий режим es6(‘use strict’); a=5; console.log(a); ошибки прошлого не работают

**ООП –**  основанная на представлении программы в виде совокупности [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), каждый из которых является экземпляром определённого [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))

John.\_\_proto\_\_=soldier;

**Динамическая типизация языка –**

**To string** -

String(4);

Console.log(typeof(“q’ + false));

**To number -**

Number(“123”);

Console.log(typeof(+’5’)); унарный плюс

parseInt(‘15px’ , 10);

**To boolean –**

Boolean(“sdasd”);

Console.log(typeof(!!’5’));

**0,’’,undefined,’null’,NaN = =false;**

Методы строк –

“строка”.slice(1, 5);

“sdasd”.length();

.toUpperCase();

.toLowerCase();

charCodeAt(i)

replace();

Метод substr() возвращает указанное количество символов из строки, начиная с указанной позиции.

Метод substring() возвращает подстроку строки между двумя индексами, или от одного индекса и до конца строки.

**Методы массивов -**

.push(); - добавление в конце

.unshift(); - добавленеи в начале

.pop(); - удаление в конце и возращает значение

.shift(); - удаление в начале

.concat(); - обьединение

.indexOf(); - поиск индекса

.join( - ); .join( и ); .join(); - обьединяет массив в строку и можно добавить разделитель

.split( - ); - обьединяет строку в массив;

.sort(); -быстрая сортировка

Math.random() \* 10; - рандомные числа от 0 до 9

Math.floor(); Math.round(); - округление числ

**Стек (last in first out) –**

Задача поиск дома подруги и возращение назад -

Var landmark = [];

Landmark.push(“мой дом”);

Landmark.push(“дорожка к дому”)

Landmark.push(“дом подруги”)

Landmark.pop(“дом подруги”)

Landmark.pop(“дорожка к дому”)

**Очередь (first in first out);**

**Действие с элементами странницы :**

Document.getElementBy … querySelectorAll(вложеность + метод for each который не доступен псевдомасивам)… querySelector ( первый элемент)

div.innerHTML = …

Все методы работают по отношению к родителю

Element.style… bgc = ‘red’, br = ‘100%’

createElement()

createTextNode

textContent(….);

classList.add ...

element.appendChild();

insertBefore();

removeChild();

replaceChild();

element.value; in inputs

element.disabled;

element.delay(5000);

eminent.fadeTo(1000);

Обьект Date –

**New Date**(Date.parse(time)).getFullYear(); - parse свойство обьекта показывает колво секунд с 1970

**ClassList.add,contains,remove, toggle,length,item;**

**Метрики :**

// const height = box.clientHeight;

// const width = box.clientWidth;

// const height = box.clientHeight;

// const width = box.clientWidth;

const height = box.scrollHeight;

const width = box.scrollWidth;

console.log(width, height);

btn.addEventListener('click', () => {

    // box.style.height = box.scrollHeight + "px";

    console.log(box.scrollTop);

});

document.documentElement.scrollTop = 0;

window.scrollBy(0, 400);

window.scrollTo(0, 400);

Get element’s positioning:

console.log(box.getBoundingClientRect().top);

Get element all styles :

const style = window.getComputedStyle(box);

console.log(style.display);

**Регулярные выражение**

Используются для удобной работы со строками. Состоит из паттерна(шаблоны, что мы ищем ) и флаги(вспомогательные символы – i не зависит от регистра, g – глобально, m – многострочности, /./ - искать или заменять любые символы ).

/\// - \ экранирующий слеш отделяющий(/,$,%);

Классы - \d – digits \D – не цифра, \w – words \W – не буква, \s –spaces \S – не пробел

let reg = /n/;

console.log(ans.**search**(reg)); - ищет первое

console.log(ans.**match**(reg)); - ищет все

console.log(reg.test(ans));

console.log(pass.replace(/./g,"\*")); - заменяет все символы

console.log(pass.replace(/-/g,"\*")); - заменяет все дефизы

let str = "Любо, братцы, любо!";

alert( str.match(/любо/gi) );

alert( "We will, we will".replace(/we/ig, "I") );

setTimeout (**рекурсивный вызов функции** внутри себя , чтобы не уверен что тело функции не перегружает ее) ,setInterval

Часть шаблона можно заключить в скобки (...)

Без скобок шаблон go+ означает символ g и идущий после него символ o, который повторяется один или более раз.

Скобки группируют символы вместе. Так что (go)+ означает go, gogo, gogogo

alert( 'Gogogo now!'.match(/(go)+/i) ); // "Gogogo"

**Делегация событий**

1. btnBlock.addEventListener('click', function (event) {

        if (event.target && event.target.tagName == "BUTTON") {

            console.log("hello");

        }

1. if (event.target && event.target.classList.contains('first')) {
2. if (event.target && event.target.matches("button.first")) {

**Обработчики событий –**

Виды - 1) через тег прописываем весь js, 2) через файл находим елемент и пишем функцию

3)**addEventListener**(“sobitie”, function(e); у обьекта e есть методы 4) forEach(f(item) + addEL

**Event.preventDefault();**

**Всплытие событий** – это когда событие происходит сначала срабатывает на вложенном элементе , потом на родителе , а потом выше и выше

Window – все окно, document окно в window которое подгружает html php картинки и тд

**Type of event :**

Click, input ( change );

‘DOMContentLoaded’

‘Loaded’

**Событие на мобилке –**

Touchstart,touchmove,touchend,touchenter,touchleave,touchcancel

**Свойства event**

console.log(e.type);

console.log(e.target.tagName == “BUTTON”);

console.log(e.target.);

console.log(e.pageX, e.pageY.);

console.log(e.touches[0].target); - mobile touches 0 finger, e.touches[0].pageX; - move by x gorizontal

console.log(e.changedTouches);

console.log(e.targetTouches);

e.code === ‘Escape’;

SEO

Возможны такие варианты:

<meta name="robots" content="index, follow"/> — в этом случае разрешена индексация страницы и ссылок.  
<meta name="robots" content="noindex, follow"/> — запрещена индексация содержимого страницы, но разрешен переход по ссылкам.  
<meta name="robots" content="index, nofollow"/> — разрешена индексация, но запрещен переход по ссылкам.  
<meta name="robots" content="noindex, nofollow"/> — запрещается и индексация, и переход по ссылкам.

**Тег каноничности** (**rel**=“**canonical**”) — это фрагмент кода HTML, который определяет основную версию для страниц с одинаковым, практически идентичным и очень похожим содержимым

Viewport for mobile

Description content

Ключевое слово **this** можно использовать в теле метода, чтобы обратиться к объекту, для которого этот метод вызывается.( текущий обьект)

Ключевое слово **this** делает методы более гибкими, позволяя добавлять один и тот же метод ко многим объектам так, чтобы он имел доступ к свойствам того объекта, для которого в данный момент вызывается.

        var human = {

            health: 'good',

            age : 20,

            speak: speak

        };

        human.speak();

        function speak (){

                console.log(this.age, this.name);

            }

В this мы можем добавлять не только свойства, но и методы.

            this.draw = function () {

            var carHtml = '<img src="car.png">';

            this.carElement = $(carHtml);

            this.carElement.css({

                position: "absolute",

                left: this.x,

                top: this.y

            });

            $("body").append(this.carElement);

        }

        };

Прототипы JavaScript — это механизм, который упрощает использование общей функциональности (то есть методов) разными объектами. У всех конструкторов есть свойство prototype, к которому можно добавлять методы; любой метод, добавленный к свойству prototype, будет доступен всем объектам, которые созданы с помощью этого конструктора.

           var Car = function (x, y) {

            this.x = x;

            this.y = y;

        };

        Car.prototype.draw = function () {

            var carHtml = '<img src="car.png">';

            this.carElement = $(carHtml);

            this.carElement.css({

                position: "absolute",

                left: this.x,

                top: this.y

            });

            $("body").append(this.carElement);

        }

        var tesla = new Car(20, 20);

        var nissan = new Car(100, 200);

        nissan.draw(tesla);

        tesla.draw(nissan);

Code review

linter - программа, которая проверяет код на соответствие стандартам в соответствии с определённым набором правил.

самодокументированном коде вам не нужно объяснять каждую строку, потому что каждый идентификатор (переменная, метод, класс) имеет четкое семантическое имя.

Code guides :

Вынесение констант вверх

Отсутствие magick numbers

Dry – don’t repeat yourself

Ненужные операции , проверки

if days == 365

DAYS\_IN\_A\_YEAR = 365

CANVAS

**Rectangular + square:**

canvas.getContext("2d");

ctx.fillStyle = "Red"; ctx.fillRect(0, 0, 100, 100);

**StrokeRect:**

ctx.strokeStyle = "DeepPink";

ctx.strokeRect(10, 10, 100, 20);

ctx.lineWidth = 4;

**Lines:**

ctx.strokeStyle = "Turquoise"; ϖ ctx.lineWidth = 4; ω ctx.**beginPath(**); ξ ctx.**moveTo**(10, 10); ψ ctx.**lineTo**(60, 60); ζ ctx.**moveTo**(60, 10); { ctx.**lineTo**(10, 60); | ctx.**stroke**();ctx.fill();

**Circles:**

ctx.lineWidth = 2;

ctx.strokeStyle = "Green";

ctx.beginPath();

ctx.**arc**(50, 50, 20, 0, Math.PI / 2, false);

ctx.stroke(); // 1/4

ctx.beginPath();

ctx.arc(100, 50, 20, 0, Math.PI, false);

ctx.stroke(); 1/2

ctx.beginPath(); 1

ctx.arc(150, 50, 20, 0, Math.PI \* 2, false);

ctx.stroke();

**Контекст вызова**

Контекст вызова – глобальный

console.log(this === window); // true

a = 37;

console.log(window.a); // 37

this.b = "MDN";

console.log(window.b) // "MDN"

console.log(b) // "MDN"

**Контект вызова функций**

1) Обычная функция *this* = window, но если use strict = undefined;

2) Контекст вызова обьекта - сам обьект;

3) *this* в конструкторах и классах - это новый экземпляр обьекта;

4) Ручная  привязка *this* call, apply, bind (биндим функцию с телом функции и новый контекст);

5) В обработчике событий контекст вызова обычной функции - это сам элемент , а в стрелочной нету контекста вызова

Классы

Классы появились в ES6 (как синтаксический сахар – удобней и не меняет поведение)

Классы – красивая обертка для функций конструкторов

Принципы ООП

**Абстракция** — отделение **концепции** от ее экземпляра

ColoredRectangular extends Rectangular {}

super(arguments); - наследование супер конструктора

использование нового обьекта один раз, не сохраняя в переменную

        new Card().render();

        const card = new Card();

        card.render();

Типы классов:

* Abstract **class**.
* Concrete **class**.
* Sealed **class**.
* Static **class**.
* Instance **class**.
* Partial **class**.

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **Choice 1** | | Abstract |
| |  | | --- | | **Choice 2** | | Concrete |
| |  | | --- | | **Choice 3** | | Prototypical |
| |  | | --- | | **Choice 4** | | Singleton |
| |  | | --- | | **Choice 5** | | Virtual |

publish/subscribe design pattern

Miscellaneous

Alt shift f – beautifier

Ctrl j

Undefined – переменная создана , но не имеет значения

Зарезервированные слова

[Scope](https://developer.mozilla.org/ru/docs/Glossary/Scope)

Garbage collector

Стек вызова функций

PUG OR JADE HTML PREPROCESSOR

Тесты js

Block scoping {}

Синхронное, ассинхроное выполнение функций(setTimeout)

Event loop просто цикл котоырй берет следующий елемент из очереди

**Замыкание** — это комбинация функции и лексического окружения, в котором эта функция была определена. Другими словами, замыкание дает вам доступ к [Scope](https://developer.mozilla.org/ru/docs/Glossary/Scope) внешней функции из внутренней функции. В JavaScript замыкания создаются каждый раз при созданиии функции, во время ее создания.

Позволяет получать доступ к локальным переменным

import {sortByDistance} from "./distance"; импортирование функций

import”vue” , import Vue from “vue” Vue.use(); взаимодействие с файлом

Шаблонные литералы заключены в обратные кавычки (` `)

*// Интерполяция в scss*

$var: 'awesome';

.#{$var}-selector {

display: flex;

}

*// Интерполяция в js*

**var** age = 25;

console.log(`I am ${age} years old`); *// I am 25 years old*

filename: `${PATHS.assets}js/[name].js`

**Оператор spread:**

Let numbers = [1,2,3];

Function log (a,b,c){cl(a);cl(b);cl(c)}

Log(…numbers);

… перед массивом чтобы его записать полностью

**Оператор rest:**