# НАЧАЛО

**Emmet**

Ставлю ! и нажимаю TAB получается базовая HTML разметка

.class>button>div

<div class="class">

<button>

<div></div>

</button>

</div>

Alert – выводит какое-то сообщение с кнопкой OK

Confirm – выводит заданное сообщение с OK и Отмена. В зависимости от того, что нажмёт пользователь – ответ либо true или false.

const result = confirm("Are you here?");

console.log(result);

Promt – позволяет пользователю ввести данные в диалоговое окно

const answer = prompt("Вам есть 18 лет?", "18");

То, что введёт пользователь – всегда СТРОКА (нельзя складывать)

Если перед promt поставить +, то будет числовое значение

const answer = +prompt("Вам есть 18 лет?", "18");

Интерполяция – значит внутри строки можно вписывать переменные.

const category = 'cat';

console.log('https://domain.com/' + category); // конкатенация (сложение)

console.log(`https://domain.com/${category}`); // интерполяция

const user = "Ivan";

alert (`Привет, ${user}`);

## ОПЕРАТОРЫ JS

console.log("arr" + " - object"); //arr - object

console.log(4 + " - object");     //4 - object

console.log(4 + +" - object");    //Nan - not a number + после сложения означает, что в кавычках числовое значение

console.log(4 + +"5");            //9

/\* ИНКРИМЕНТ И ДИКРИМЕНТ \*/

let incr = 10,

    decr = 10;

   // префиксный - делает сразу операцию и возвращает результат СРАЗУ

console.log(incr++); //11

console.log(decr); //9

// ЕСЛИ СПЕРВА ВЫПОЛНЯЕМ ОПЕРАЦИЮ А ПОТОМ ВЫЗЫВАЕМ КОНСОЛЬ – то работает

incr++;

decr--;

    //ПОСТФИКСНЫЙ   - СНАЧАЛА ВОЗВРАЩАЕТ СТАРОЕ ЗНАЧЕНИЕ, чтобы получить значений переменной её нужно вызвать ещё раз

/\* ++incr;

--decr; \*/

/\* console.log(incr++); //11

console.log(decr--); //9 \*/

/\* console.log(incr);

console.log(decr);  \*/

console.log(--incr);

console.log(++decr);

console.log(incr);

console.log(decr);

console.log(5%2); // вернёт 1. т.к. это остаток.

Сравнение

console.log(2\*4 == 8);

выдаст **TRUE**

Сработает даже если 8 записать строкой

console.log(2\*4 == '8');

Для строгого равенства нужно ===

console.log(2\*4 === '8');

False (nfr как число сравнивается со строкой, если кавычки убрать, то снова true

Логические операторы И, ИЛИ (&&, ||)

Опертор && возвращает правду только, если оба выражения равны

|| возвращает когда оба верны или хотя бы 1.

const isChek = true,

        isClose = true;

    console.log(isChek&&isClose);

Результат true

Отрицание !

    const isChek = false,

        isClose = false;

    console.log(isChek||!isClose);

вернёт true, Так как второе было сперва false, но восклицательный знак дал отрицыние и оно превратилось в true. 1 из 2 условий выполнено.

# GIT система контроля версий

В рабочей папке

git init

git config --local user.name "Sasha"

git config --local user.email [hfgghf@gmail.com](mailto:hfgghf@gmail.com)

**Глобально:**

git config --global user.email [hfgghf@gmail.com](mailto:hfgghf@gmail.com)

**КОММИТ – контрольная точка**

**Статус репозитория:**

Git status

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

.jshintrc

JS.docx

index.html

js/

~$JS.docx

Добавить файл в отслеживаемые:

Git add-A (ВСЕ ФАЙЛЫ)

Git add main.css – (только 1 конкретный файл)

**Добавить файлы в КОММИТ**

git commit -a -m"5 commit"

-a = значит все файлы. M – месседж для сохранения (то, что в кавычках).

**Посмотреть все логи**

git log

**ГИТХАБ**

Заходим на гитхаб. Открываем новый репозиторий. <https://github.com/proektion1?tab=repositories>

git branch -M main

git remote add origin <https://github.com/proektion1/projekt.git> //адрес репоз

git push -u origin main

Потом просто с добавление нового файла или изменения

Git add-A – добавление

git commit -a -m"first commit" - коммит

git push – ЗАПУШИТЬ НА ГИТХАБ!

**СКАЧАТЬ НА ВТОРОЙ КОМП с ГИТХАБА СВОЙ ПРОЕКТ**

1. **Сперва в терминале переходим в нужную папку cd “** **C:\Users\SUPER GAMER\Desktop\work\”**
2. **Git clone** [**https://github.com/proektion1/projekt.git**](https://github.com/proektion1/projekt.git) **project2**
3. **Новый проект скачается в папку work, внутри папки project2**

**ЕСЛИ работаете с нескольких устройств – нужно обновлять папки**

**git pull**

**выйти из терминальнеых сообщений**

**Ctrl + С (2 раза)**

**:wq!**

# УСЛОВИЯ:

const num = 50;

 if (num<49) {

     console.log('Error');

 } else if (num>100) {

     console.log('More');

 } else {

     console.log ('Ok');

 }

**Тернарный оператор – в его работе учавствуют 3 аргумента.**

 const ff = 50;

 (ff === 50) ? console.log ('Ok') : console.log('Error');

**Бинарный аргемент 4 + 4. Унарный: +4.**

const yy = 50;

 switch (yy) {

     case 49:

        console.log('Неверно');

        break;

    case 100:

        console.log('Неверно');

        break;

     case 50:

        console.log('В точку!');

        break;

    default:

        console.log('В другой раз!');

        break;

 }

# ЦИКЛЫ

let num = 50;

while (num <55) {

    console.log(num);

    num++;

}

**ИЛИ**

let num = 50;

do {

    console.log(num);

    num++;

}

while (num<55);

**или**

for (let i = 1; i <8; i++) {

    console.log(num);

    num++;

}

**До 6 шага, потом прекратить**

for (let i = 1; i <8; i++) {

    if (i === 6) {

        break;

    }

    console.log(i);

}

**Исключить 6, и продолжить**

for (let i = 1; i <8; i++) {

    if (i === 6) {

       continue;

    }

    console.log(i);

}

# Функции, стрелочные функции

// создаём функцию

function showFirstMessage (text) {

    console.log(text);

}

showFirstMessage("Hello, world!"); // запускаем функцию и передаём ей параметр

Если создадим переменную внутри функции – она не выйдет из неё.

**Она называется локальной!**

function showFirstMessage (text) {

    console.log(text);

    let num = 20;

}

showFirstMessage("Hello, world!"); // запускаем функцию и передаём ей параметр

console.log(num);

**Глобальная может перезаписаться**

let num = 20; //глобальная переменная

function showFirstMessage (text) {

    console.log(text);

    num = 10; // перезаписываем переменную

}

console.log(num); // выведет 20

showFirstMessage("Hello, world!"); // запускаем функцию и передаём ей параметр

console.log(num); // выведет 10, так как запустилась функция, и в ней перезаписалась переменная

**Локальная и глобальная переменная**

let num1 = 20; //глобальная переменная

function showFirstMessage (text) {

    console.log(text);

    let num1 = 10; // локальная переменную

}

console.log(num1); // выведет 20

showFirstMessage("Hello, world!"); // запускаем функцию и передаём ей параметр

console.log(num1); // выведет 20, так как внутри функции была ОБЪЯВЛЕНА (не перезаписана) локальная переменная,

                // за её прееделами будет работать глобальная она будет только внутри функции

**ЗАМЫКАНИЕ ФУНКЦИИ – сама функция со всеми внешними переменными, которые ей доступны.**

function calc (a, b) {

    return (a + b); // возвратит сложение

}

console.log(calc(4, 3));

console.log(calc(5, 6));

function ret () {

    let num = 50;

    return num; // вернёт нам во внешний мир значение переменной

}

const anotherNum = ret(); // забираем результат из функции

console.log(anotherNum); // даст 50

****

1. Деклорейшн - создаётся сразу. Можно к ней обращаться в начале кода,

а сама функция может быть в конце. Браузер сперва считывает все такие функции.

В примерах выше, как раз она!

2. Экспрешн – только тогда, когда до неё доходит код

const logger = function() {

    console.log("Hello");

};

logger();

3. Стрелочная функция

const caalc = (a, b) => a + b;

# Методы и свойства строк и чисел

**Методы – вспомогательные функции**

**Свойства – вспомогательные значение**

**Свойство (подсчёт символов в переменной):**

const str = "test";

console.log(str.length);

**//4**

# Callback function

**Функция, которая должна быть выполнена после того, как другая завершилась.**

function learnJS(lang, callback) { //объявляем функцию с 2 аргументами

console.log(`Я учю: ${lang}`);

callback();                     //выполняем 2 функцию только после console.log

}

learnJS('Javascript', function() {      //запускаем функцию с 2 аргументами

    console.log('Я прошёл этот урок!'); //2 аргумент - функция с текстом

});

# Объекты

const options = {

    name: 'test',

    width: 1024,

    height: 1024,

    colors: {

        border: 'black',

        bg: 'red',

    }

};

console.log(options.name);

//delete options.name;

console.log(options);

for (let key in options) {

    if (typeof(options[key]) === 'object') {

        for (let i in options[key]) {

        console.log(`Свойства ${i} имеет значение ${options[key][i]}`);

        }

    } else {

        console.log(`Свойства ${key} имеет значение ${options[key]}`);

    }

}

**Кол-во объектов в массиве**

const options = {

    name: 'test',

    width: 1024,

    height: 1024,

    colors: {

        border: 'black',

        bg: 'red',

    }

};

console.log(Object.keys(options));

**Значения:** ${options[key]}

**Вывести ключи объектов:** Object.keys

**Вывести значения:** Object.values

**Ключ-значение:** Object.entries

**Подсчитать кол-во объектов в массиве** Object.keys.length

console.log(Object.keys(miniBD).length); //4

**Деструктуризация Объекта - вынос вложенных объектов из объектов (чтобы не писать некрасивый код:**

console.log(`Свойство ${i} иммеет значение ${miniBD[key][i]}`);

const miniBD = {

    type: 'short',

color: {

        line: 'red',

        topii: 'blue',

        general: 'red',

    }}

const {line, topii} = miniBD.color;

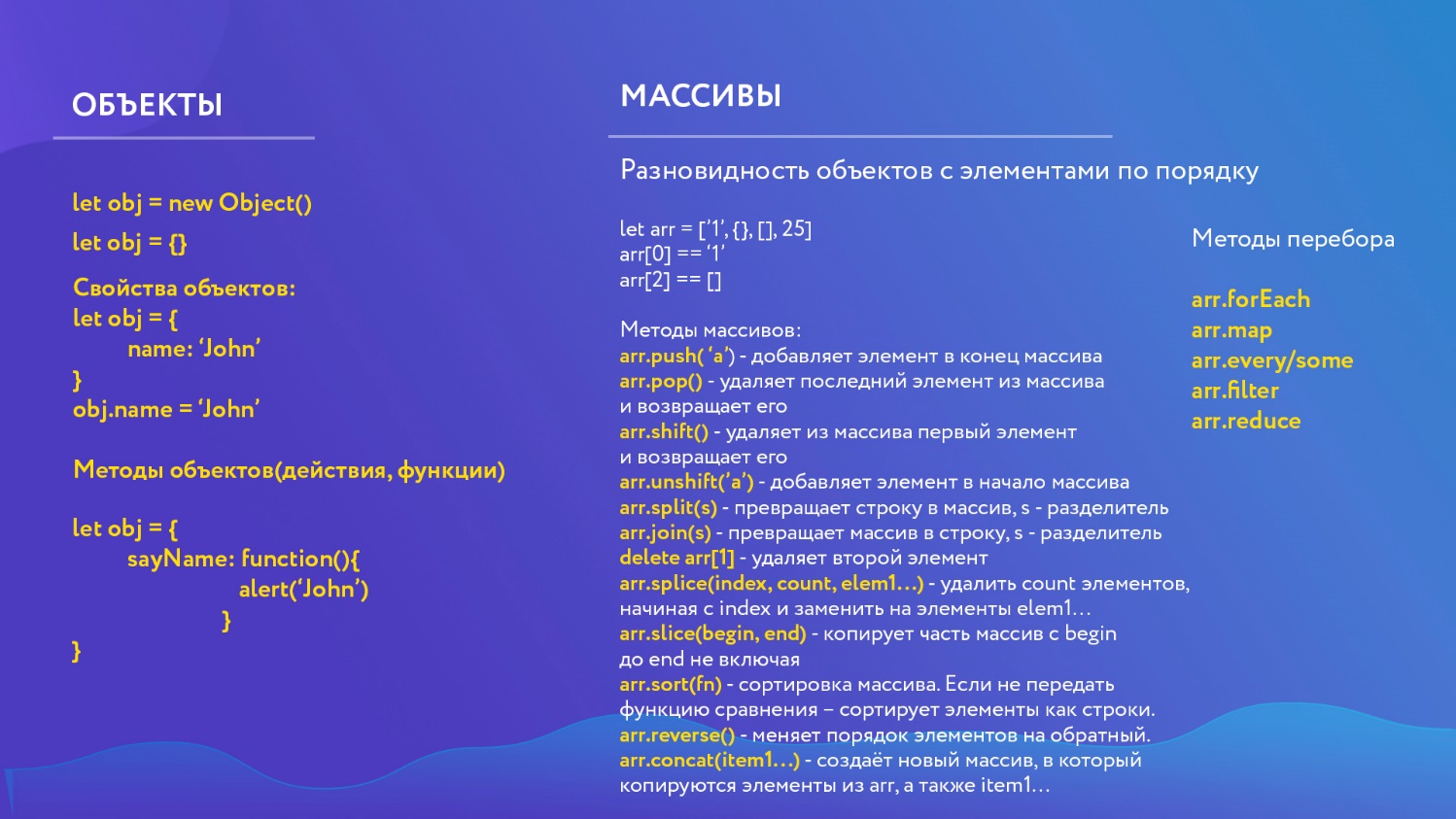
console.log(line);

**ПЕРЕБОР МАССИВОВ**

**FOR IN – перебирает индексы**

**FOR OF – перебирает значения (массив можно перебрать, объект нет)**

**В for of и for in можно сделать continue и break. В FOREACH – нельзя.**

****

const arr = [1, 2, 3, 6, 8];

arr.forEach(function (item, i, arr) {   // CALLBACK функция. forEach для каждого элемента выполняется

    //item - сами значения в массиве,

    //i - порядковый номер, arr - значения массива (все)

    console.log(`${i}: ${item} внутри массива ${arr}`);

});

/\* 0: 1 внутри массива 1,2,3,6,8

1: 2 внутри массива 1,2,3,6,8

2: 3 внутри массива 1,2,3,6,8

3: 6 внутри массива 1,2,3,6,8

4: 8 внутри массива 1,2,3,6,8 \*/

arr.pop(); // удаляет последний элемент массива

arr.push(10); //добавляет в конец массива

// Запрашиваем данные и превращаем в массив

const str = prompt("", "");

const products = str.split(", "); //запятая - разделитель элементов (НУЖНО ЗНАТЬ)

console.log(products); // выведет массив

console.log(products.join('; ')); // из массива в строку ; - разделитель

**СОРТИРОВКА**

// sort использует алгорит быстрой сортировки

// сортировка (КАК СТРОКИ - по алфавиту, по 1 символу)

const produc1ts = [2, 11, 4, 9, 12, 22, 222];

produc1ts.sort();

console.log(produc1ts);

//11, 12, 2, 22, 222,  4, 9

// сортировка по заданным условиям

const produc11ts = [2, 11, 4, 9, 12, 22, 222];

produc1ts.sort(compareNum);

function compareNum(a, b) {

    return a - b;

}

console.log(produc11ts);

//2, 4, 9, 11, 12, 22, 222

19 ПЕРЕДАЧА ПО ССЫЛКЕ  
В простых типах данных присваивание происходит легко.

В объектах (объекты, массивы, функции) идёт не присваивание, а передача по ссылке

const obj = {

    a: 5,

    b: 1,

};

const copy = obj; // ЗДЕСЬ ПЕРЕДАЁТСЯ ЗНАЧЕНИЕ ПО ССЫЛКЕ (это 12 строка)

                  // в  copy кладётся ссылка

console.log(copy); //{ a: 5, b: 1 }

copy.a = 10;

console.log(copy); //{ a: 10, b: 1 }

console.log(obj); //{ a: 10, b: 1 } СТРАНННО!!! СМ 12 строку

Для присваивания или получения данных нужно использовать другие методы.

//ДЕЛАЕМ КЛОНИРОВАНИЕ ЧЕРЕЗ ФУНКЦИЮ

function copppy(PARAMETR) {         // СОЗДАЕМ ФУНКЦИЮ

    let objCopy = {};               // создаём пустой объект, чтобы туда записать значения

    for (let key in PARAMETR) {     // Перебираем ключи (свойства) в пеереданном PARAMETR

       objCopy[key] = PARAMETR[key];//значение PARAMETR передаем в пустой objCopy

    }

    return objCopy;                 //возвращаем objCopy из функции во внешнюю среду

}

const numbers = {

    a: 2, // свойство а которое равно 2

    b: 5,

    c: {

        x: 7,

        y: 4

    }

};

const newNumbers = copppy(numbers); //КЛОНИРУЕМ!!!

console.log(newNumbers);// ПРОВЕРЯЕМ { a: 2, b: 5, c: { x: 7, y: 4 } }

newNumbers.a = 8888;    //МЕНЯЕМ значение свойства

console.log(newNumbers); //{ a: 8888, b: 5, c: { x: 7, y: 4 } }

console.log(numbers);   // { a: 2, b: 5, c: { x: 7, y: 4 } }

**Когда мы так клонируем объекты, нужно запомнить, что есть глубокие и поверхностные копии. Когда нет вложенных объектов или массивов – поверхностные отрабатывают хорошо (как у нас), но если в массиве были вложенные объекты, то у вложенных снова будет передача по ссылке!**

/\* Добавляем к объекту ещё 1 объект \*/

const lulu = {

    asa: 43,

    tt: 65,

};

const add = {

    d: 17,

    i: 20

};

console.log(Object.assign(lulu, add));

/\* Добавляем к объекту ещё 1 объект \*/

const lulu = {

    asa: 43,

    tt: 65,

};

const add = {

    d: 17,

    i: 20

};

//console.log(Object.assign(lulu, add)); СКЛАДЫВАЕМ

const gggg = Object.assign(lulu, add); // ЗАВИСИМЫЙ lulu поверхностный

const gggg1 = Object.assign({}, add); //НЕЗАВИСИМЫЙ поверхностный

/\* Самый простой способ клонирвоать объект: \*/

const namePer = Object.assign({}, add); // add - объект, который нужно скопировать

/\* 3 способ - ТОЛЬКО с МАССИВОМ

Метод slice() возвращает новый массив,

содержащий копию части исходного массива. \*/

const oldArray = ['a', 'b', 'c'];

const newArray = oldArray.slice();

newArray[1] = 'dsfsdf';

console.log(newArray); //[ 'a', 'dsfsdf', 'c' ]

console.log(oldArray); //[ 'a', 'b', 'c' ]

# ООП – начало

const solder = {

    health: 400,

    armor: 100,

    sayHello: function() {

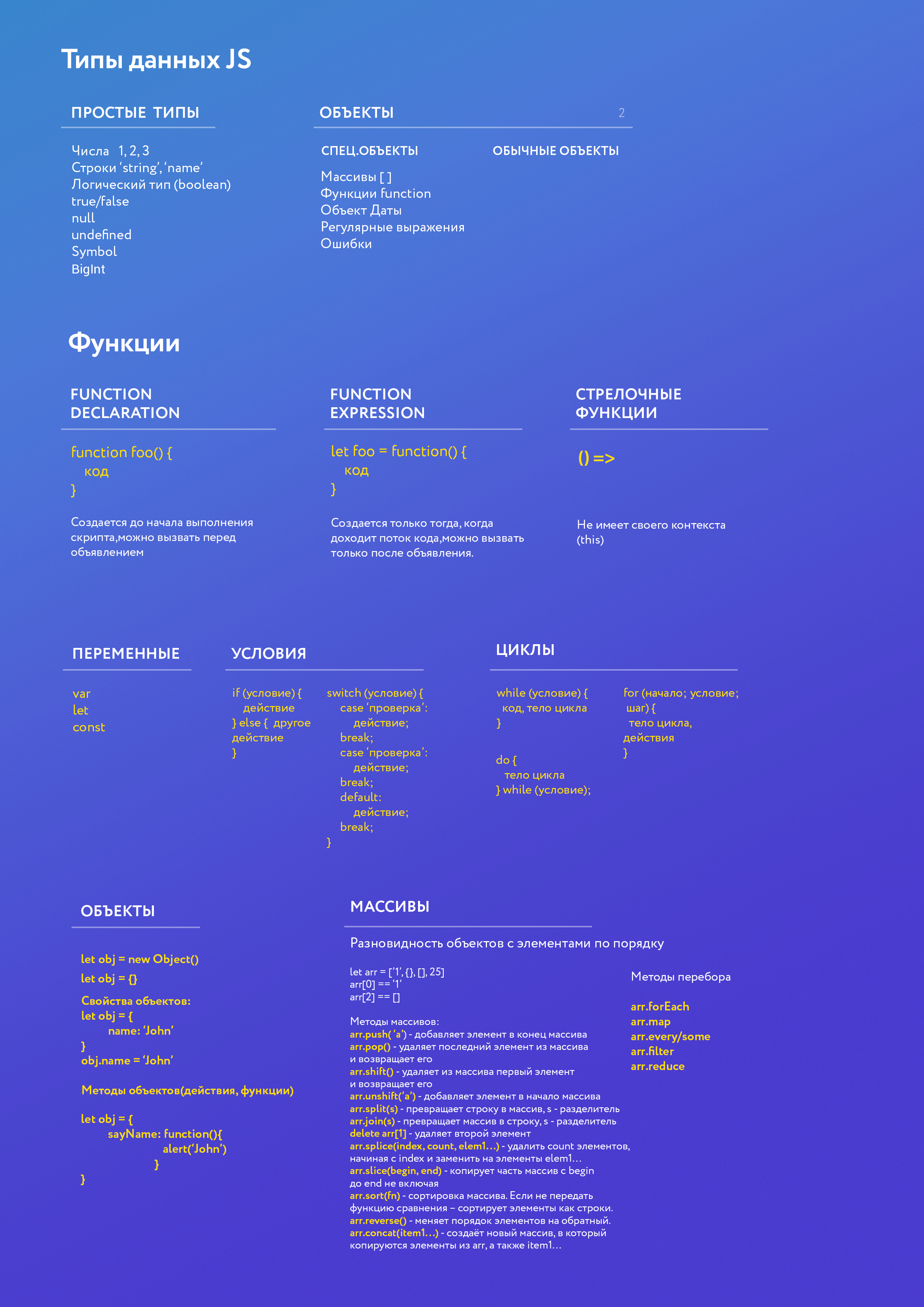
        console.log("Hello!");

    }

};

const jonh = Object.create(solder);

jonh.sayHello(); // Hello



# Динамическая типизация

Это возможность одного типа данных превращаться в другой (число может стать строкой).

Статическая типизация – число всегда остается числом.

/\* Переводим в строковое значение

1 способ. (страый)\*/

console.log(typeof(String(null))); //string

console.log(typeof(String(4)));     //string

/\* 2 способ - конкотенация \*/

console.log(typeof(5+ ''));        // string

const num = 5;

console.log("https://vk.com/catalog/" + num); //https://vk.com/catalog/5

const fontSize = 26 + 'px';

/\* Превращаем в число \*/

/\* 1 способ \*/

console.log(typeof(Number('4'))); //number

/\* 2 способ - унарный плюс \*/

console.log(typeof((+'54')));   //number

let answer = +prompt("Hello", "");

/\* 3 способ \*/

console.log(typeof(parseInt('15px', 10))); //number 10 - вернуть в 10-ричной системе

console.log(parseInt('15px', 10));          //15 - начинается с числа, буквы отбрасывает

console.log(parseInt('ф15px', 10));         //NaN - т.к. начинается с буквы

/\* Превращаем в булиновые значение \*/

/\* 0, '', null, undefined, Nan; - всё это FALSE \*/

/\* TRUE \*/

/\* 1 способ) \*/

let switcher = null;

if (switcher) {             // if false - значит не выполнится

    console.log('Workinh');

}

switcher = 1;

if (switcher) {

    console.log('Workinh'); //Workinh

}

/\* 2 способ \*/

console.log(typeof(Boolean('4'))); //boolean

console.log((Boolean('4')));        //true

/\* 3 способ \*/

console.log(typeof(('4')));      //string

console.log(typeof(('!4')));     //string

console.log(4);                 //4

console.log('!4');              //!4

console.log(typeof((!4)));      //boolean

console.log(typeof((!!'4')));   //boolean

console.log(!4);                //false

console.log(!!4);               //true

# ЗАДАЧИ САМОСТОЯТЕЛЬНО

//Какое будет выведено значение: let x = 5; alert( x++ ); ? 6

let x = 5;

alert( x++ ); //5 делает сложение потом

alert( x++ ); //6

let xx = 11;

alert( ++xx ); //12 сразу делает сложение

//Чему равно такое выражение: [ ] + false - null + true ?

console.log([ ] + false);         //false

console.log([ ] + false - null);   //Nan

console.log([ ] + false - null + true); //Nan

console.log(typeof([] + false)); //string

console.log(typeof([ ] + false)); //string

//Пустой массив - пустая строка

//Что выведет этот код: let y = 1; let x = y = 2; alert(x); ?

let y = 1;

let x = y = 2;  // так не пишут

console.log(x); //2

//Чему равна сумма [ ] + 1 + 2?

console.log([] + 1 + 2); //12

//[] - даст пустую строрку. дальнейшей сложение идёт, как сложение строк

console.log([] + 6);     //6

console.log([]);         //[]

//Что выведет этот код: alert( "1"[0] )?

console.log( "1"[0] );     //1 - т.к.[0] - это индекс строки, т.е. 1

console.log( "1"[1] ); //undefined, так как только 0 индексов

//• Чему равно 2 && 1 && null && 0 && undefined ?

console.log(2 && 1 && null && 0 && undefined); //null

console.log(2 && 1); //1

console.log(2 && 1 && null); //null

/\* && - замыкается на ложном выражении, || замыкается на правде

после него код перестанет выполняться

Идёт слева направо\*/

/\* •    Есть ли разница между выражениями? !!( a && b ) и (a && b)? \*/

console.log(!!( 1 && 2 ) === (1 && 2)); // FALSE т.к. !! всегда вернёт boolean false

/\* •    Что выведет этот код: alert( null || 2 && 3 || 4 ); ? \*/

console.log(null || 2 && 3 || 4); //3

/\* && выполняется первее || т.к. из-за приоритета выполнения

1. 2 && 3 даст 3

2. null || 3, даст 3.

3. далее 3 || 4 , а || замыкается на правде \*/

console.log(3 || 4); //3

console.log(3 && 4); //4

/\* •    a = [1, 2, 3]; b = [1, 2, 3]; Правда ли что a == b ? \*/

const a = [1, 2, 3];

const b = [1, 2, 3];

console.log(a==b); //false

// a и b - разные хранилища информации с одинаковыми значениями.

// Мы сравниваем не элементы, а сами хранилища, они разные

/\* •    Что выведет этот код: alert( +"Infinity" ); ? \*/

console.log( +"Infinity" ); //Infinity + означает, что в строке числовое значение

console.log(typeof(+"Infinity"));//number

/\* •    Верно ли сравнение: "Ёжик" > "яблоко"? \*/

console.log("Ёжик" > "яблоко"); //false идёт

посимвольное сравнение "Ё не равно я"

/\* •    Чему равно 0 || "" || 2 || undefined || true || falsе ? \*/

console.log(0 || "" || 2 || undefined || true || falsе); //2

/\* 0 - false

"" - false

2 - true

2 || undefined  || замыкается на true \*/

# DOM Получение элементов со страницы

/\* СТАРЫЕ МЕТОДЫ \*/

/\* 1. Получаем объект по ID \*/

const box =  document.getElementById('box');

console.log(box);

/\* 2. Получаем элементы по имени тега \*/

const btns = document.getElementsByTagName('button');

//получили псеводомассив

//То есть у него есть числовые свойства (индексы) и свойство length.

//У них нет свойств, и методов массивов, таких как forEach, join, slice и др.

const btn1s = document.getElementsByTagName('button')[1];//получили конкретную кнопку

console.log(btns);      //вывести все кнопки

console.log(btns[1]);   //вывести конкретную кнопку

console.log(btn1s);     //вывести конкретную кнопку

/\* 3. Получаем элемент по имени класса \*/

const circles = document.getElementsByClassName('circle'); // псевдомассив

console.log(circles);

/\* НОВЫЕ МЕТОДЫ \*/

/\* Получаем css селектор (класс, вложеность, атрибут и т.д.) \*/

## [Основные виды селекторов](https://learn.javascript.ru/css-selectors" \l "osnovnye-vidy-selektorov)

Основных видов селекторов всего несколько:

* \* – любые элементы.
* div – элементы с таким тегом.
* #id – элемент с данным id.
* .class – элементы с таким классом.
* [name="value"] – селекторы на атрибут (см. далее).
* :visited – «псевдоклассы», остальные разные условия на элемент (см. далее).

**Селекторы можно комбинировать, записывая последовательно, без пробела:**

* .c1.c2 – элементы одновременно с двумя классами c1 и c2
* a#id.c1.c2:visited – элемент a с данным id, классами c1 и c2, и псевдоклассом visited

## [Отношения](https://learn.javascript.ru/css-selectors" \l "otnosheniya)

В CSS3 предусмотрено четыре вида отношений между элементами.

Самые известные вы наверняка знаете:

* div p – элементы p, являющиеся потомками div.
* div > p – только непосредственные потомки

Есть и два более редких:

* div ~ p – правые соседи: все p на том же уровне вложенности, которые идут после div.
* div + p – первый правый сосед: p на том же уровне вложенности, который идёт сразу после div (если есть).

/\* НОВЫЕ МЕТОДЫ \*/

/\* Получаем css селектор (класс, вложеность, атрибут и т.д.) \*/

// document.querySelectorAll Позволяем применять forEach

const hearts = document.querySelectorAll('.heart');

console.log(hearts);

hearts.forEach(item=> {

    console.log(item);

})

const jneHeart = document.querySelector('.heart'); // Позволяет получить самый 1 элемент

console.log(jneHeart); // самый первый heart

# Действия с элементами на странице DOM

const box =  document.getElementById('box');

box.style.backgroundColor = 'blue'; //меняем цвет

box.style.width = '500px';          //меняем ширину

const btns = document.getElementsByTagName('button');

const circles = document.getElementsByClassName('circle');

btns[1].style.borderRadius = '100%';//сглаживаем углы

circles[0].style.backgroundColor = 'red';

***Прописываем несколько свойств ксс***

box.style.cssText = 'background-color: blue; width: 900px;';

Меняем свойства у всех объектов

const hearts = document.querySelectorAll('.heart');

hearts.forEach(item => {

    item.style.backgroundColor = 'blue';

});

Создать элемент и вставить на страницу

const div = document.createElement('div');

div.classList.add('black');

//black уже описан в стайл.ксс

//div дали класс black

document.body.append(div);

// Берем body и внутрь его вставляем созданный div

/\* Добавляем без определения переменной \*/

//Берем wrapper и добавляем div

//document.querySelector('.wrapper').append(div);

//wrapper.append('div');

wrapper.prepend(div); //Вставить перед wrapper

div.innerHTML = "<h1>Hello World</h1>"; //Вставляем HTML

div.textContent = "Hello";  //Вставляем текст

div.innerHTML = "<h1>Hello World</h1>";

div.insertAdjacentHTML('afterbegin', '<h2>hi</h2>');

/\* 'beforebegin': до самого element (до открывающего тега).

'afterbegin': сразу после открывающего тега  element (перед первым потомком).

'beforeend': сразу перед закрывающим тегом element (после последнего потомка).

'afterend': после element (после закрывающего тега). \*/

hearts[0].before(div); // вставить перед первым элементом

hearts[0].after(div); // вставить после первого элементом

circles[0].remove(); // удалить элемент

hearts[0].replaceWith(circles[0]); //заменить элемент hearts на circles

УСТАРЕВШИЕ КОНСТРУКЦИИ

wrapper.appendChild(div); // ДОБАЛЯЕМ

wrapper.insertBefore(div, hearts[0]); //вставить div перед hearts[0].

wrapper.removeChild(hearts[1]); //удалить элемент

wrapper.replaceChild(circles[0], hearts[1]); //вместо hearts ставим circles