1. Переменные

Переменная – учвсток памяти компьютера (браузера), который имеет имя и содержит некоторое значение. Значение переменной может изменяться.

"use strict";//директива которая указывает скриптовому файлу,

//что мы работаем в современном режиме, !использовать всегда!

//прописывать в начале скриптового файла

let number = 5; // = - присваивание //let - переменная изменяемая

const leftBorderWidth = 1; // цифра не может быть первой в названиии

//переменной,нельзя использовать зарезирвированные слова.

number = 10;

console.log(number);

const obj = { //прямых констант в js небывает. что такое прямая константа

    a:50

};

obj.a = 10;

console.log(obj);

**Строковые операции**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Запись операции*** | ***Название*** | ***Возвращаемый результат,***  ***выполняемое действие*** | ***Пример использования*** |
| a+b | Конкатенация  (склейка) | Возвращает результат «склейки» двух операндов | let a = ‘machine’;  console.log(‘coffee’ + a);  **coffeemachine**  console.log (‘coffee’ + ‘cup’);  **coffeecup** |
| a+=b | Присваивание с конкатенацией | «склеивается» операнд с указанной строкой  Значение опеанда изменяется | let a=’перекати’;  a+=’поле’;  console.log (a);  **перекатиполе** |

**Арифметические операции**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Запись операции*** | ***Название*** | ***Возвращаемый результат,***  ***выполняемое действие*** | ***Пример использования*** |
| a+b | сложение | Возвращает сумму двух операндов | let a=5;  console.log (a+2);  **7** |
| a-b | вычитание | Возвращает разность двух операндов | let a=5;  console.log (7-a);  **2** |
| a\*b | умножение | Возвращает произведение двух операндов | let a=5;  console.log (a\*2);  **10** |
| a/b | деление | Возвращает частное двух операндов | let a=7;  console.log (a/2);  **3.5** |
| a%b | деление по модулю (остаток от деления) | Возвращает остаток от деления двух операндов | let a=11;  console.log (a%4);  **3** |
| -a | изменение знака | Возвращает операнд с изменённым знаком | let a=5;  console.log (-a);  **-5** |
| --a | предекремент | 1.уменьшает операнд на 1;  2.возвращает операнд  **Значение операнда изменяется** | let a=5;  console.log (--a);  **4**  console.log(a);  **4** |
| ++a | преинкремент | 1.увеличивает операнд на 1;  2.возвращает операнд  **Значение операнда изменяется** | let a=3;  console.log (++a);  **4**  console.log(a);  **4** |
| a-- | постдекремент | 1.возвращает операнд;  2.уменьшает операнд на 1;  **Значение операнда изменяется** | let a=3;  console.log (a--);  **3**  console.log(a);  **2** |
| a++ | постинкремент | 1.возвращает операнд;  2.увеличивает операнд на 1;  **Значение операнда изменяется** | let a=3;  console.log (a++);  **3**  console.log(a);  **4** |
| a+=c | присваивание со сложением | увеличивает операнд на указанное значение;  **Значение операнда изменяется** | let a=3;  a+2;  console.log(a);  **5** |
| a-=c | присваивание с вычитанием | уменьшает операнд на указанное значение;  **Значение операнда изменяется** | let a=3;  a-2;  console.log(a);  **1** |
| a\*=c | присваивание с умножением | увеличивает операнд на указанное количество раз;  **Значение операнда изменяется** | let a=3;  a\*2;  console.log(a);  **6** |
| a/=c | присваивание с делением | уменьшает операнд в указанное количество раз;  **Значение операнда изменяется** | let a=10;  a/2;  console.log(a);  **5** |

**Операции сравнения**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Запись операции*** | ***Название*** | ***Возвращаемый результат,***  ***выполняемое действие*** | ***Пример использования*** |
| a<b | Меньше | Возвращает **true**, если первый операнд меньше второго | let a=5;  console.log (a<8)  **true**  let a=5;  console.log (a<5)  **false**  let b=’book’;  let c=’notepad’  console.log (b<c)  **true**  console.log (c<b)  **false** |
| a>b | Больше | Возвращает **true**, если первый операнд больше второго | let a=5;  console.log(a>8);  **false**  console.log(a>5);  **false**  console.log(a>2);  **true** |
| a<=b | Меньше либо равно | Возвращает **true**, если первый операнд меньше, либо равен второму | let a=5;  console.log (a<=8);  **true**  console.log (a<=5);  **true**  console.log (a<=2);  **false** |
| a>=b | Больше либо равно | Возвращает **true**, если первый операнд больше либо равен второму | let a=5;  console.log(a>=8);  **false**  console.log(a>=5);  **true**  console.log(a>=2);  **true** |
| a==b | Равно(нестрогое равенство) | Возвращает **true**, если первый операнд равен второму | let a=5;  console.log(a==8);  **false**  console.log(a==5);  **true**  console.log(a==’5’);  **true** |
| a!=b | Не равно | Возвращает **true**, если операнды не равны | let a=5;  console.log(a!=8);  **true**  console.log(a!=5);  **false**  console.log(a!=’5’);  **false** |
| a===b | Тождественное равенство (строгое равенство) | Возвращает **true**, если операнды равны и значением и типом | let a=5;  console.log(a===8);  **false**  console.log(a===5);  **true**  console.log(a===’5’);  **false** |
| a!==b | Не тождественно | Возвращает **true**, если операнды не равны значением либо типом | let a=5;  console.log(a!==8);  **true**  console.log(a!==5);  **false**  console.log(a!==’5’);  **true** |

**Логические операции**

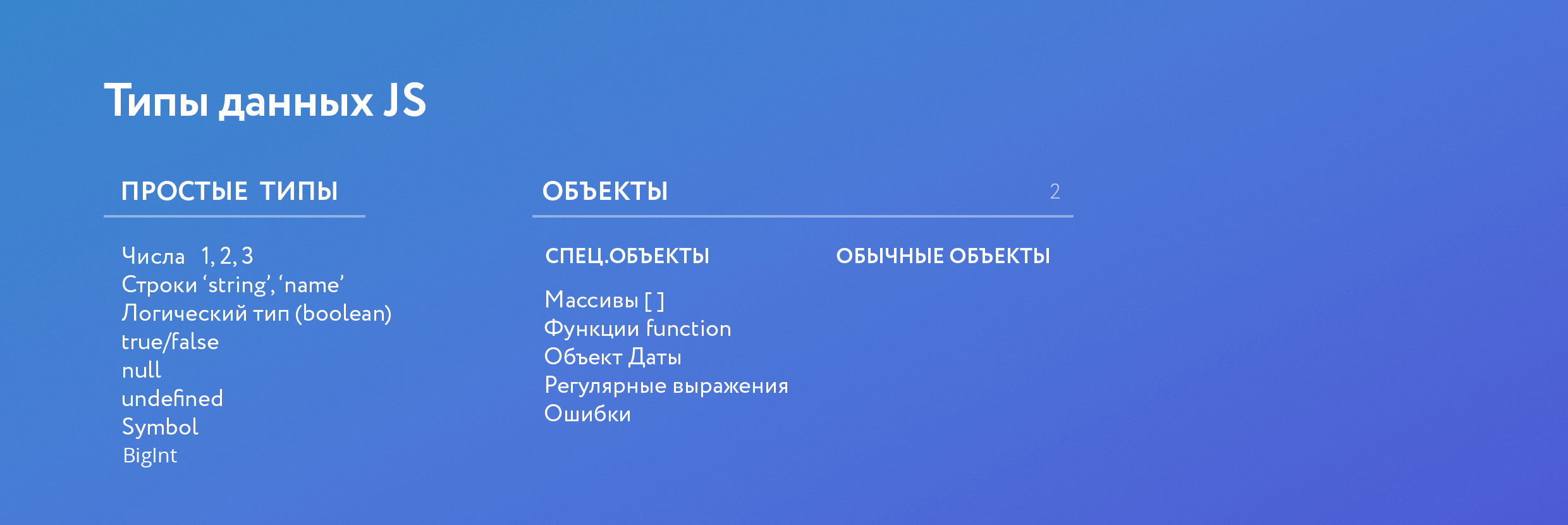
|  |  |  |  |
| --- | --- | --- | --- |
| ***Запись операции*** | ***Название*** | ***Возвращаемый результат,***  ***выполняемое действие*** | ***Пример использования*** |
| a&&b | Логическое **И** | Возвращает true, если операнды истинны | let name = ‘Alex’;  let age = 25;  console.log ((name == ‘Sergey’) && (age > 18));  **false**  console.log ((name == ‘Alex’) && (age > 50));  **false**  console.log ((name == ‘Alex’) && (age > 18));  **true** |
| a||b | Логическое **ИЛИ** | Возвращает true, если один из операндов истинен | let distance = 2000;  let sum = 700;  console.log((distance<1000) || (sum>500));  **true**  console.log((distance<1000) || (sum>1000));  **false** |
| !a | Логическое **НЕ** | Если операнд ложен –**true**  Если операнд истинен - **false** | let temperature = 35;  let zharko = (temperature>30);  console.log (!zharko);  **false** |

**Преобразование типов и проверка результата**

Преобразование из строкового в числовой тип

|  |  |  |
| --- | --- | --- |
| Способ | Возвращаемое значение, выполняемое действие | Пример использования |
| Функция  **parseInt(знач)** |  |  |
| Функция  **parseFloat(знач)** |  |  |
| Функция  **Number(знач)** |  |  |
| Применение любой операции, допустимой только для числовых значений |  |  |
|  |  |  |
|  |  |  |

1. Типы данных



Undefined – что-то существует, но этого нет (стоит холодильник, но он пустой )

BigInt – работа с большими числами. (ограничение с обычными числами. Мы не можем задать число, которое больше чем 2 в 53 степени)

"use strict";

//тип данных - число

let number = 4.6; //числа, даже дробные

console.log(4/0);//infinity

console.log('string'\*9);//nan - не число

//типданных - строка

const persone = "5";

//логичесий тип данных (boolean)

const bool = true;

//тип анных null И undefined

console.log(something);

let und;

console.log(und);

//тип данных - объект

const obj = {// это объект

    name: "john",// свойства объекта

    age: 25,// свойства объекта

    isMarried: false// свойства объекта

};

console.log(obj.name); // обратились к значению свойства name  объекта

//массив это частный случай объекта, о не является типом данных

console.log(obj["name"]);//конструкция в квадратных скобках

//используется для хранения данных идущих по порядку

let arr = ['plum.png', 'orange.jpg', 'apple.bmp', 6, {}, []];

console.log(arr[1]);// в программировании исчисление начинается с 0

//[Running] node "e:\3.Projects JS\first\tempCodeRunnerFile.js"

//orange.jpg

1. Простое общение с пользователем

"use strict";

//alert('hello');

//const result = confirm('are you here?');

//console.log(result);

//происходит запись результата

//const answer = prompt("Вам есть 18?", "17");

//console.log(answer); // ответ записывает от пользователя, тип данных строка

//вся информация от пользователя птип данных является строка,

//если не преобразован тип данных программистом

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

//console.log(typeof(answer)); в этом случаи будет number тип данных

const answers = [];

//answers[0]=prompt ('как ваше имя', '');

//answers[1]=prompt ('ваша ффамилия', '');

//answers[2]=prompt ('сколько вам лет', '');

//document.write(answers);

console.log(typeof(answers));

console.log(typeof(null));//ошибка, но такая вот особенность

4.Интерполяция

Это значит, что мы можем вставлять внутри строки значение переменной.

const category = 'toys';

//console.log('https://someurl.com/' + category); // конкоцинация (https://someurl.com/toys)

console.log(`https://someurl.com/${category}/5`);//не забывать про апострофы

const user = 'Ivan';

console.log(`Privet, ${user}`);

5.Операторы

console.log('arr' + '- object'); // конкоцинация, строки сложатся и правратятся

//в большую строку -  arr- object;

console.log(4 + '- object'); // тоже самое

console.log(4 + +'5');// 9, унарный плюс +"5", использует один аргумент

//икримент и дикримент

let incr = 10,

    decr = 10;

    incr++;//увеличение на единицу(постфикснаяя форма записи)

    decr--;//уменьшение знач на ед.(постфикснаяя форма записи)

   // ++incr;//(префиксная форма записи)

// --decr;//(префиксная форма записи)

    console.log(incr); //11 тоже самое будет и при префиксной форме

    console.log(decr); // 9

    console.log(incr++); //10

    console.log(decr--); //10

    /\*в постфиксной фоме возвращается старое значение,

     а после произвоит действие\*/

     console.log(5%2); // 1 - выводит остаток

     console.log(2\*4 == 8); //true сравнение происходит по значению, а не по типу данных

     // если сравнивать по типу данных то

     console.log(2\*4 === '8'); // false, строгое сравнение, сравнивает и по тиру данных

     /\* && - оператор и - работает когда два либо более значения являются правдивыми

        || - оператор или - работает когда хотябы одно значение является правдивым

      \*/

    const isChecked = true;

          isClose = false;

          console.log(isChecked || isClose); // выдает true

          const isChecked = true;

          isClose = false;

          console.log(isChecked && isClose); // выдает false, true выдаст, когда оба значения будут true

          // ! - оператор отрицания

           console.log(2 + 2 \* 2 === 8);//false

           console.log(2 + 2 \* 2 != 8); // true

           console.log(2 + 2 \* 2 !== '6');//true

callback – исполнение строго заданное, пока не выполнится заданная функция, втрая выполнятся не будет

'use strict';

function first() {

    //do something

    setTimeout(function() { //эмуляция задержки на 0,5с

        console.log(1);

    }, 500);

}

function second() {

    console.log(2);

}

first();

second();

function learnJS(lang, callback) {

    console.log(`я учу ${lang}`);

    callback();

}

function done(){

    console.log('я прошёл этот урок');

}

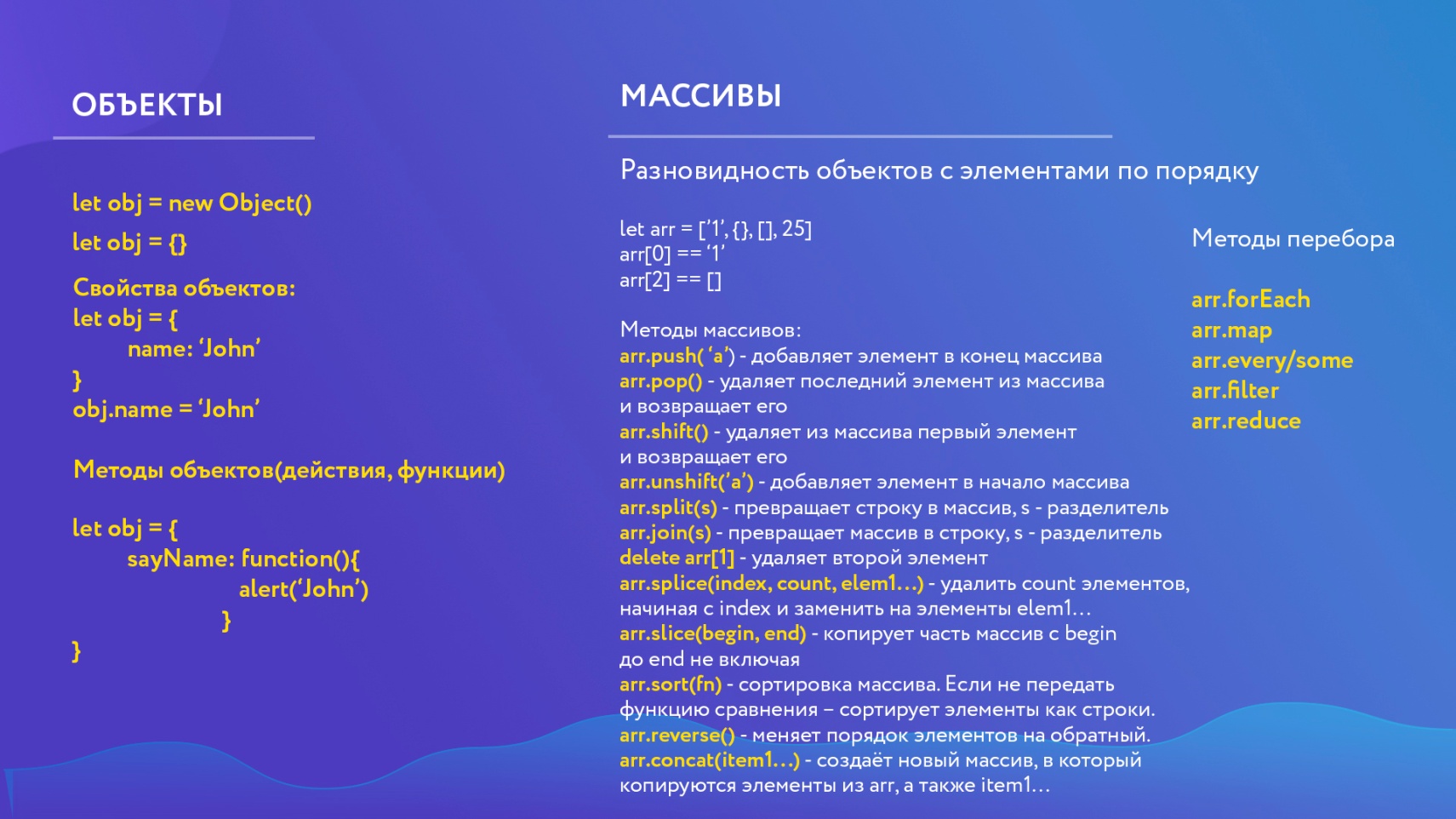
learnJS('JAvascript', done);

//круглых скобок нет, потому что мы не вызываем функцию,

// а передаем, что бы она была использована



Объекты, деструктуризация объектов



'use strict';

//обекты деструктуризация объектов

const options = {

    name: 'test',

    width: 1024,

    height: 1024,

    colors:{

        border: 'black',

        bg: 'red'

    },

    //методы - это действия которые может совершать объект

    //а этот метод мы создаем сами

    makeTest: function(){

        console.log('Test');

    }

};

options.makeTest();

//деструктуризация

const {border, bg} = options.colors;

//мы вытащили кусочки вложеные в свойства в качестве отдельнй переменной

console.log(border);

//console.log(Object.keys(options).length); // получение количества элементов внутри объекта

//это встроенныеметод

//ЗАПОМНИТЬ!!!

//console.log(options.name);

//delete options.name;//опертор удаления

//console.log(options);

//{ width: 1024, height: 1024, colors: { border: 'black', bg: 'red' } }

//без строчки тест

//прием счетчика

/\*let counter = 0;

for (let key in options) { //роемся в свойствах

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

        for (let i in options[key]){

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

            counter++;

        }

    } else {

        console.log(`Свойства ${key} имеет значение ${options[key]}`);//key это name width и т.д

        //таким обазом мы будем получать значение этого ключа которые перебирается в этой итерации

        //значений цикла

        counter++;

    }

}

console.log(counter);

\*/

//ИТОГ

/\*

чтобы перебрать мы можем использовать конструкцию for in

у объектов бывают встроенные методы  встроенные свойства

так же мы можем и создавать методы

 \*/

021 массивы и псевдомассивы

"use strict";

//Массивы и псевдомассивы

const arr = [1, 2, 3, 6, 8];

arr.sort(compareNum);

console.log(arr);

function compareNum(a,b) {

    return a - b;

}

//почитать алгоритмы

//arr[99] = 0;

//сonsole.log(arr.length);

//стостоит из последнего индекса в массиве + 1

//arr.forEach(function(item, i, arr){

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

//});

//arr.pop();//удление последнего элемента с массива

//[ 1, 2, 3, 6 ]

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

//[ 1, 2, 3, 6, 8, 10 ]

//console.log(arr);

//for (let i = 0; i < arr.length; i++){

  //  console.log(arr[i]);

//} //перебор элементов

//for (let value of arr) {

   // console.log(value);

//}//второй пособ перебора элементов но:

/\*

работает с массивоподобными сущностями

 \*/

 const str = prompt('', '');

 const products = str.split('', '');

 products.sort();//сортирует элементы как строки

 console.log(products.join('; '));

Основы ООП

//Основы ООПб прототипн-ориентированное наследование

let str = 'some';

let strObj = new String(str);

// console.log(typeof(str));

// console.log(typeof(strObj));

console.dir([1, 2, 3]);

const soldier = {

    health: 400,

    armor: 100,

    sayHello: function() {

        console.log('hello');

    }

};

// const john = {

//     health:100

// };

// устаревший формат

// john.\_\_proto\_\_ = soldier;

// console.log(john.armor);

// console.log(john.sayHello);

// Object.setPrototypeOf(john, soldier);

// john.sayHello();

const john = Object.create(soldier);

john.sayHello();

Передача по ссылке или по значению, оператор spread

'use strict';

//передача по ссылке или по значению, оператор spread

// let a = 5,

//     b = a;

//     b = b +5;

//     console.log(b);

//     console.log(a);

    //const obj = {

       // a: 5,

       // b: 1

    //};

    //const copy = obj;//передаёт ссылку, мы не скопировали новый объект

    //а передает именно ссылку на существующий объект

//copy.a = 10;

//console.log(copy);

//console.log(obj);

function copy(mainObj) {

    let objCopy = {};

    let key;

    for (key in mainObj) {

        objCopy[key]=mainObj[key];

    }

    return objCopy;

}

const numbers = {

a: 2,

b: 5,

c: {

    x: 7,

    y: 4

 }

};

const newNumbers = copy(numbers);

newNumbers.a = 10;

newNumbers.c.x = 10;//измениться везде

console.log(newNumbers);

console.log(numbers);

/\*{ a: 10, b: 5, c: { x: 10, y: 4 } }

{ a: 2, b: 5, c: { x: 10, y: 4 } }

потому что создана поверхностная копия

\*/

const add = {

    d: 17,

    e: 20

};

const clone = Object.assign(numbers, add);

clone.d = 225;

// console.log(add);

// console.log(clone);

const oldArray = ['a', 'b', 'c'];

const newArray = oldArray.slice();//метод позволяющий скопировать старый массив

newArray[1] = 'asdfsdfsdfdfs';

console.log(newArray);

console.log(oldArray);

//четвертый способ создания поверхностной копии

// spread оператор разворота, разворачивает структуру и превращает ее

// в набор данных

const video = ['youtube', 'vimeo', 'rutube'],

    blogs = ['wordpess', 'livejounal', 'blogger'],

    internet = [...video, ...blogs, 'vk', 'facebook'];

    console.log(internet);

    function log(a , b , c) {

        console.log(a);

        console.log(b);

        console.log(c);

    }

    const num = [2, 5, 7];//информация пришелшая с сервера

    //нам нужно ередать ее в функцию

    log(... num);

//еще один способ

const array = ['a', 'b'];

const newAarray = [...array];

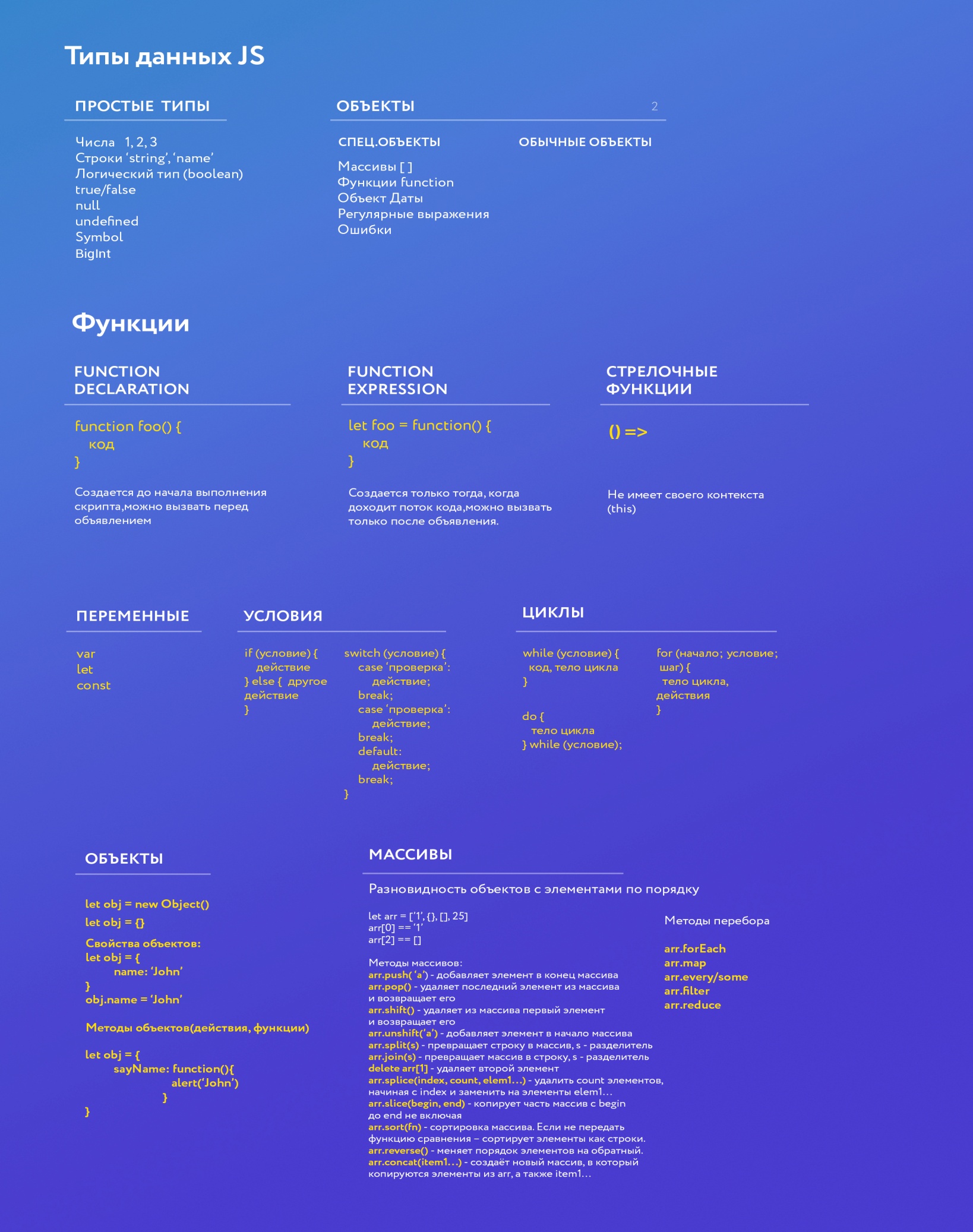
const q = {

    one: 1,

    two: 2

};

const newObj = {...q};

Динамическая типизация данных

'use strict';

//отдавлваем ошибки в своем коде

//breakpoints

//ДИНАМИЧЕСКАЯ ТИПИЗАЦИЯ

//преобразование типов данных

//TO STRING

//1

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

console.log(typeof(String(5)));

//2 конкоцинация

console.log(typeof(5 + ''));

//слоение строк или сложение строк с чем-то

//со сложением со строкой у нас получается строка

const num  = 5;

console.log("htttps://vk.com/catalogue/" + num);//устаревшая часть

const fontSize = 26 + 'px';

//создается строка которая потом может использоваться в стилях

//TO NUMBER

//1

console.log(typeof(Number(4)));

//2 унарный плюс +

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

//3 parseInt

console.log(typeof(parseInt('15px', 10)));

let answ = +prompt('hello', '');

//всё что мы получаем от пользователя будет типом данных строка

//значения из промтов, данные из тегов и т.д.

//TO BOOLEAN

//1

//0, '', null, undefined, NaN;

//всё это будет превращаться в false,

//строка с пробелом ' ' - является непустой строкой.

let switcher = null;

if (switcher){

    console.log('working...');

}

switcher = 1;

if (switcher){

    console.log('working...');

}

//2

console.log(typeof(Boolean(4)));

//3

console.log(typeof(!!"44444"));

Вопросы к собесу

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

**Всё из-за постфиксной формы записи, сразу нам выведит х, а затем только его увеличит, если запиcать префиксная ++х, то нам выведет уже увеличенное число на единицу**

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

**Когда работаем с пустым массвом он преобразуется в пустую строку, когда конкатинируем какой-то тип данных со строкой, получаем строку**

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

**Последовательное присваивание**

* Чему равна сумма [ ] + 1 + 2?

**12 сложение идёт как обычная строка**

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

**К Каждому элементу строки можно обратиться по его индексу.**

**Выведет 1**

* Чему равно 2 && 1 &&null&& 0 &&undefined ?

**оператор & запинается на лжи, когда оператор & обрабатывает все аргументы, он идёт слева на право, сразу смотрит 2, это правдивое выражение в лог. Контектсе, 1 тоже, null – дает false,**

* Есть ли разница между выражениями? !!( a&& b ) и (a && b)? False

Два восклицательных знака возвращает булиновое выражение не поэтому разница есть

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

Сразу выполняется операор с более высоким приоритетом это &, затем ||, 2 && 3, будет возвращать последнее значение т.е. 3,

Теперь смотрим сравнение null с результатом предыдущего сравнения 3, **И запинается на лжи, а ИЛИ на правде**, то выведет тройку

Далее 3 или 4, или запинается на правде, поэтому выводится 3-ка

* a = [1, 2, 3]; b = [1, 2, 3]; Правда ли что a == b ? ответ нет

потому что это две разные переменные, которые содержат одинаковые данные

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

Но потипу данных тут будет число, так как стаит унарный плюс, которые преобразует строковый тип данных в числовой

* **Верно ли сравнение: "Ёжик" > "яблоко"? false**

**Будет по символьное сравнение. Необходимо открыть таблицу с символами Unicode и посмотреть какой символ где распологается, а вот если ёжик, в нижнем регистре, то будет true, про Unicode стоит у кого-нибудь уточнить**

* Чему равно 0 || "" || 2 || undefined || true || falsе ?

**0 влогическом контексте false Пустая строка тоже false, 2 true, поэтому выведет 2**

**Меотоды получения элементов со страницы**

//методы получения элементов со страницы

//1. getElementbeID

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

// ID может быть только один на странице

console.log(box);

//2.getElementsByTagName

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

console.log(btns);

//тут мы получили не один элемент а псевдомассив,

//script.js:11 HTMLCollection(5) [button, button, button, button, button]

/\*для того чтобы плучить отдельно элементо,

нужно указать индекс искомого элемента

1. способ - const btns = document.getElementsByTagName("button")[1];

формат определения переменной, записывая htmlcollection

2. способ - console.log(btns[1])

тут мы используем один кокретный элемент

естесственно мы получим кнопку вторую, не забывай про нумерацию с 0

Если в структуре будет лиш один элемент, то все равно создастся

массив с одним элементом, при этом редактировать сам элемнт будет невоможно

нужно всегда указывать индекс\*/

//3 способ

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

console.log(circles);

//4 способ

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

//помещаем css селектор

console.log(hearts);

hearts.forEach(item => {

    console.log(item);

});

//5 способ

const oneHeart = document.querySelector('.heart');

console.log(oneHeart);

**Действия с элементами на странице(пересмотреть!)**

Закомментировать строку ctr+/

'use strict';

//действия с элементами на странице

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

    btns = document.getElementsByTagName('button'),

    circles = document.getElementsByClassName('circle'),

    hearts = document.querySelectorAll('.heart'),

    oneHeart = document.querySelector('.heart'),

    wrapper = document.querySelector('.wrapper');

    console.dir(box); //просмотр в качестве объекта

// box.style.backgroundColor = 'blue';//изменение цвета в box

// box.style.width = "500px";

//необходимо передавать все в виде строки

//инлайн стили имеет самый высокий приорите

btns[1].style.borderRadius = '100%';

//указываем инекс, чтобы точно обратиться к элементу

//ошибочный вариант

// circles.style.backgroundColor = 'red';

/\*

в консоли выдаст ошибку по причине того, что пытаемся обратиться не к

конткретному элементу, а к псевдомассиву, а он ничего не знает об объекте style

это просто структура, которая содержит элементы.

Для того чтобы все работало, необходимо четко указать элемент,

над которым будут производиться действия

\*/

//верный варент

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

//назначаем несколько инлайн стилей

box.style.cssText = 'background-color: blue; width: 500px';

// box.style.cssText = `background-color: blue; width: ${num}px `;

//такая структура полезно для динамических преобразований

// for (let i =0; i < hearts.length; i++){

//     hearts[i].style.backgroundColor = 'green';

// }

/\*запускаем новый цикл, указываем что  переменный итератор раве нулю, перебирая

элементы в массивах чаще будем указывать 0, т.к. нумерация массивах идет

с 0 индекса и перебираются по порядку

далее указывается, что наш цикл буде работать до тех пор

у нас остануться количество элементов внутри этого псевдомассива

в свойстве length указаны элементы и цикл будет

рабоать пока не переберет все количество элементов в этом свойстве.

теперь чтобы что-то сделать с каждым элементом внутри обрааемся к псевдомассиву

чтобы получить какой-то элемент в квадратных скобках ставим переменный итератор

чтобы постпенно получать элементы, начиная с 0 и до n-го, далее вносим

изменения

\*/

//следующий метод

hearts.forEach(item=> {

item.style.backgroundColor = 'blue';

});

//сдесь в коллбэке, который передается в  foreach второй аргумент обозначает

//номер по порядку,а третий имеет ссылку на тот массив, который мы перебираем

//изучение основных методов для работы с элементами страницы

//создаем переменную, обращаемся к document  и используем метод сreateElement

//в круглых скобках пишем названием тэга, который мы создаем

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

//всё элемент создан и он существует только внутри JS

//на станице он никак не появится

//так же можно создавать текстовые узлы

//это элементы без оболочки тэга

//const text = document.createTextNode('тут был я');

div.classList.add('black');

//методы для добавления элементов перед и после других элементов

// document.body.append(div);

//append - добавляет в конце родитяля какой-то элемент

wrapper.append(div);

//так как мы внесли wraper в переменные запись

//document.querySelector('.wrapper') не нужно

//прочто пишем wrapper

//старый стиль

// wrapper.appendChild(div);

// wrapper.prepend(div);

// prepend добавляет элемент перед роителем

// черный блок будет над сердечками

// hearts[0].before(div);

// hearts[0].after(div);

//старая форма записи

// wrapper.insertBefore(div, hearts[1]);

//методы по удалению элементов со страницы

// circles[0].remove();

//удаляет первый кружочек

//старая форма записи

// wrapper.removeChild(hearts[1]);

//метод замены одного элемента другим

//hearts[0].replaceWith(circles[0]);

//старая форма

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

//вписываем текст в наш div

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

//метод работает со всей html структурой

//второй вариант

// div.textContent = 'hello';

//только с текстом

// div.insertAdjacentHTML('beforebegin','<h2>Hello</h2>');

//вставляет html код перед элементом

//над черным квадратом пявилась надпись

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

//вставляет html код в начало элементом

/\*будет вставлено в div black\*/

// div.insertAdjacentHTML('beforeend','<h2>Hello</h2>');

//вставляет html код в конце элемента

// div.insertAdjacentHTML('afterend','<h2>Hello</h2>');

//вставляет html код после элемента