* + 1. strict mode
    2. типы данных
    3. способы объявления функций
    4. прототипное наследование
    5. промисы, async/await
    6. всплытие событий
    7. замыкание
    8. call, apply, bind
    9. garbage collector
    10. == and ===
    11. Spread & rest operators
    12. Map and WeakMap, Set and WeakSet
    13. Функции высшего порядка

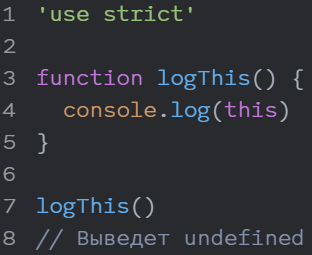
1. **strict mode**

• введен в ES5;

• применяется ко всему скрипту или к отдельным функциям, применение в контексте фигурных скобок приведет к ошибке;

• выйти из него после вхождения нельзя, аля «no use strict»

• другое поведение this: this больше не ссылается на глобальный объект (например window)



1. **типы данных**
   * + 1. **Typeof**

Всего в JS **8** типов данных:  
примитивы: number, BigInt, string, bool, null, undefined, symbol;  
объекты: object.

• Разница между null и undefined:

null – для неизвестных значений;

undefined – для не присвоенных значений.

• Оператор typeof для извлечения типа данных из переменной (возвращает строку в которой написан тип данных)

Пример: let x = 5; typeof x; // “number”

typeof null // “object”! Ошибка в языке

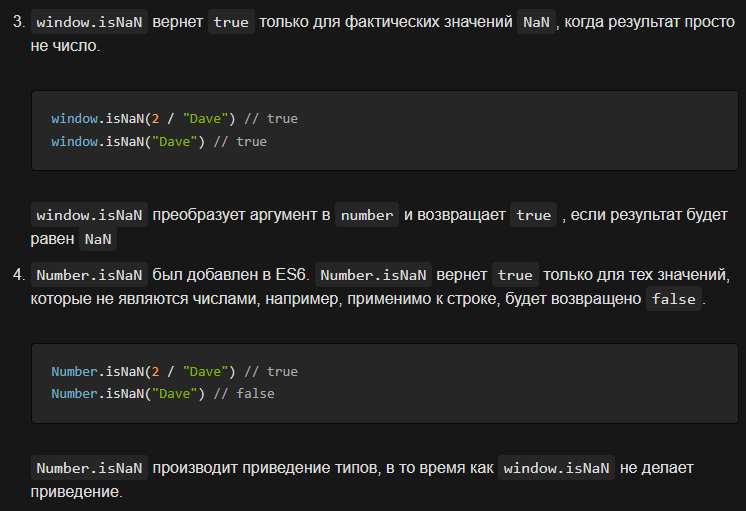
typeof NaN // “number”! // Не число, которое является числом

**typeof NaN**

1. Никогда не равен сам себе

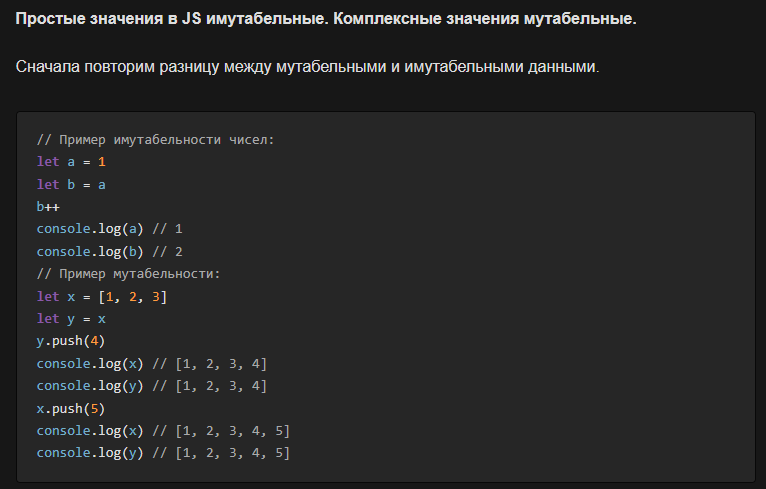
NaN === NaN //false

1. Проверка на NaN:



1. typeof NaN всегда возвращает “number”
   * + 1. **Значение и ссылка**

Пример мутабельности / имутабельности



* + - 1. **Преобразование и сравнение типов**

**Примитивы**

Преобразование типов может быть:  
• явным – целенаправленное преобразование одного типа к другому:

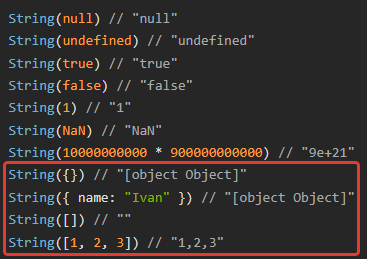
String(123) // ”123”

• не явным – преобразование типов происходит без явного указания команд на это

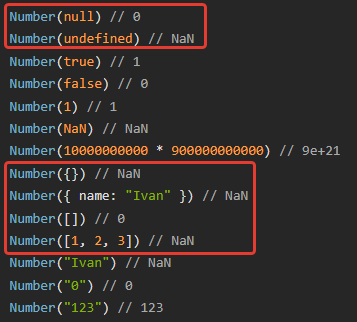
“2” + “2” = “22” //конкатенация строк

Преобразовать любой тип данных можно только к 3м типам:

• string:



• number:



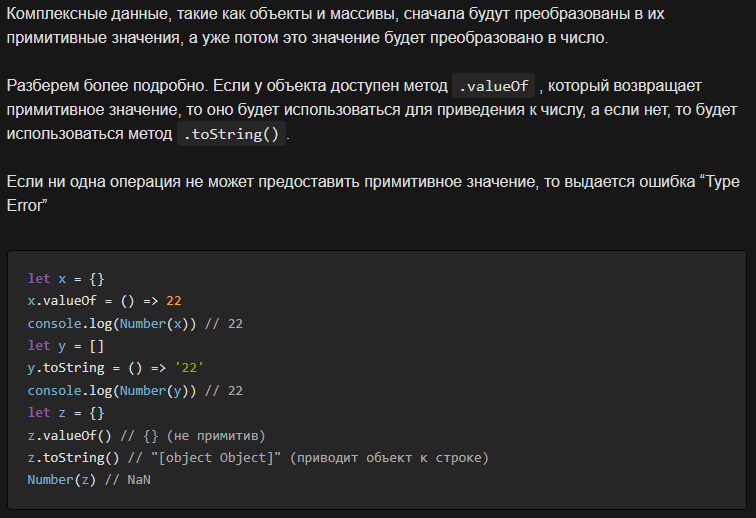
• bool

Приведение следующих типов данных к bool всегда даст false, в противном случае true:

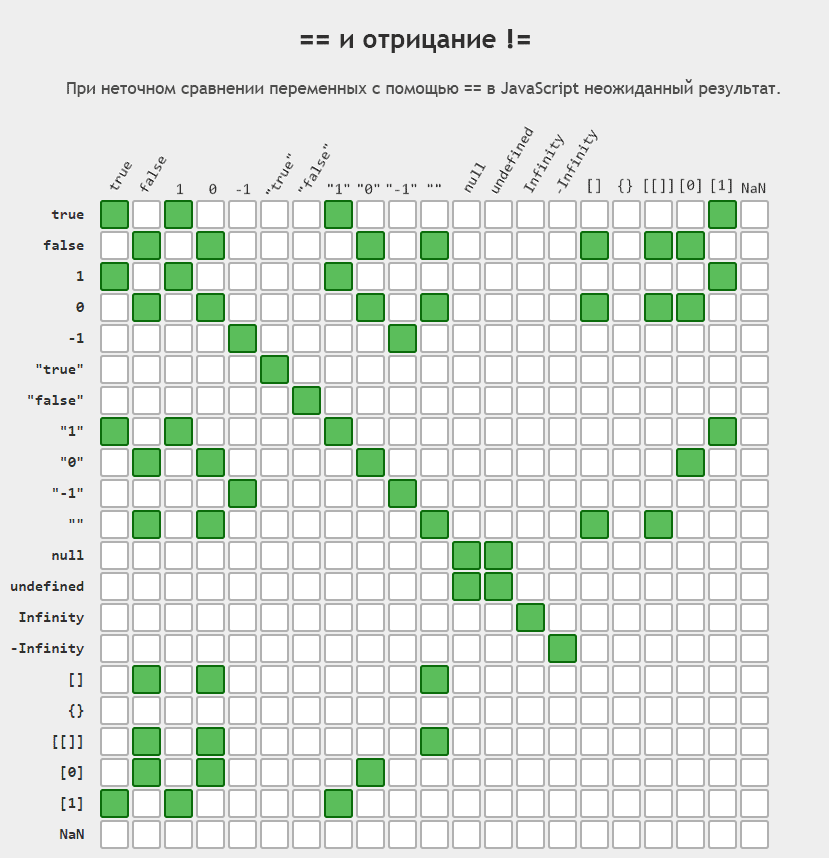
null, undefined, “”, 0, NaN; //falsy типы

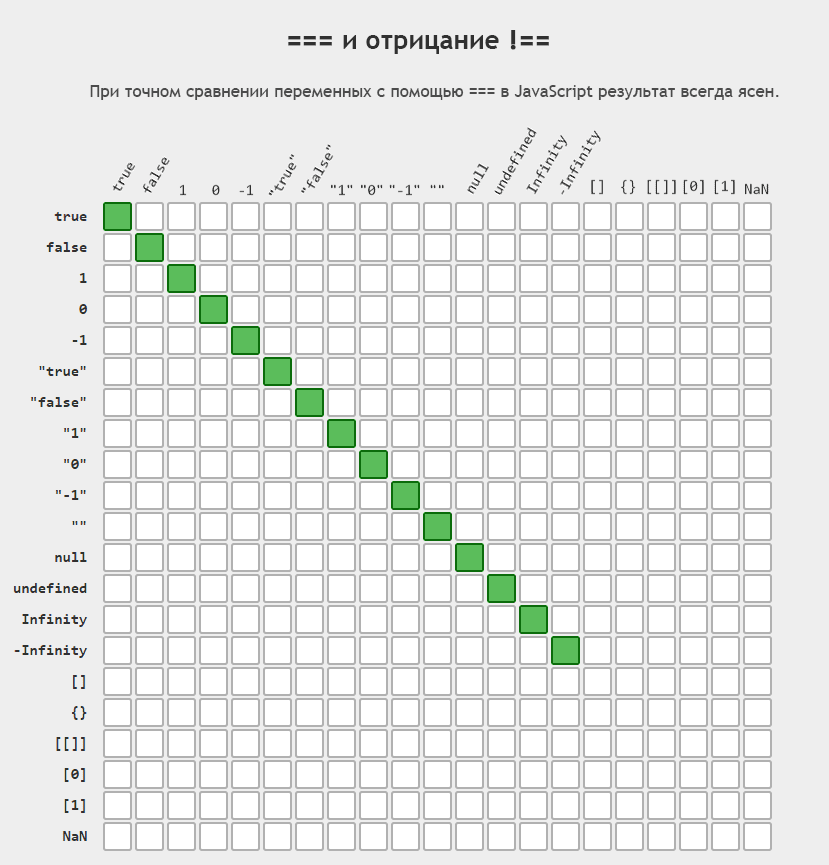
Неявное приведение между строкой и числом:

**Комплексные данные (массивы, объекты, функции)**



* + - 1. **Строгое сравнение (===) и сравнение с приведением типов (==)**

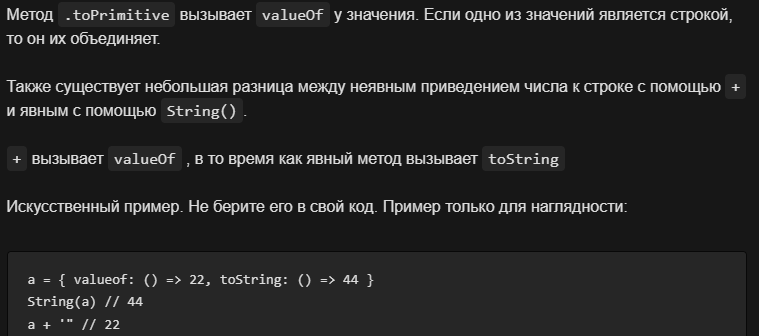




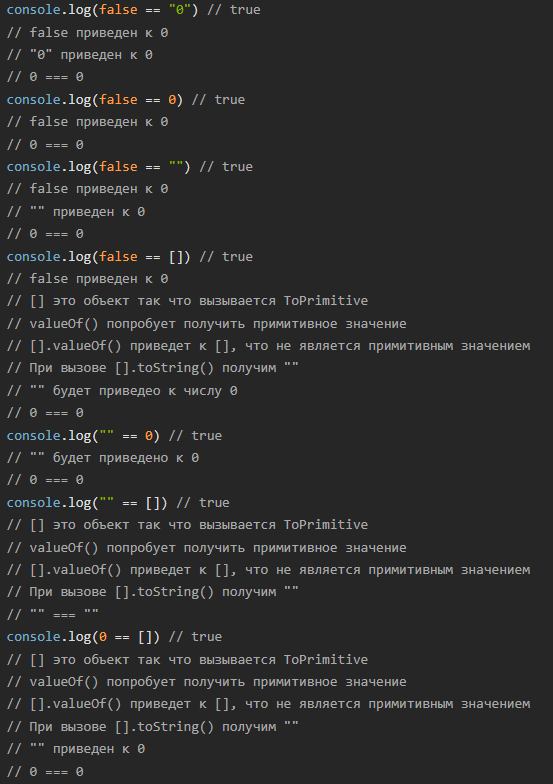
* + - 1. **Особенности**

1. Откуда у примитивов типа “string” есть методы вроде .toUpperCase?

Ответ: в таком случае вокруг примитива создается обертка в виде объекта, у которого как раз есть методы. После выполнения инструкции обертка удаляется и у нас снова остается примитивное значение.



* + - 1. **Примеры**

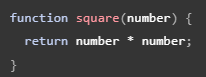




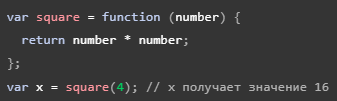
console.log("false" == false) // false  
// Оператор сравнения вызывает численное преобразование  
// Левый операнд "false" преобразуется в NaN  
// Правый операнд true станет 0  
// NaN === 0 (вернет false)  
  
console.log(null == "") // false  
// Оператор == обычно вызывает численное преобразование, но не в случае с null  
// null == null и null == undefined возращает true, а все остальные случаи вернут false  
  
console.log(!!"false" == !!"true") // true  
// Оператор !! конвертирует строки "false" и "true" в булевые значения   
// Получаем true == true, т.к. "false" не пустая строка (вернет true)  
  
console.log(["x"] == "x") // true  
// Оператор == вызывает численное преобразование у массива  
// Метод массива valueOf возвращает сам массив. Этот результат игнориуется, т.к. не является примитивом  
// Далее вызывается метод массива toString, который конвертирует ["x"] в "x"  
// "x" == "x" (вернет true)  
  
console.log([] + null + 1) // "null1"  
// Оператор + вызывает численное преобразование массива  
// Метод массива valueOf возвращает сам массив. Этот результат игнориуется, т.к. не является примитивом  
// Далее вызывается метод массива toString, который конвертирует [] в ""  
// "" + null (вернет "null")  
// "null" + 1 (вернет "null1")  
  
console.log([1, 2, 3] == [1, 2, 3]) // false  
// В данном примере преобразование не происходит, т.к. оба массива одного типа  
// Оператор == сравнивает объекты по ссылке, а не по значению  
// Данные массивы являются двумя разными экземплярами  
// Поэтому [1, 2, 3] == [1, 2, 3] вернет false

1. **способы объявления функций**

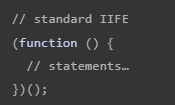
**•** function declaration statement (FDS)



**•** function definition expression (FDE)



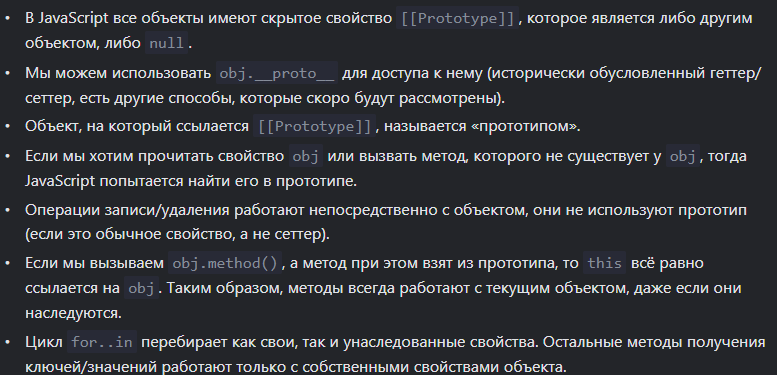
• immediately invoked function expression (IIFE)



• array functions



1. **Прототипное наследование**



Проверка наличия свойства в объекте:

• prop in obj – проверяет наличие свойства в объекте, если не находит, то обращается к obj.\_\_proto\_\_ и так по всей цепочке прототипов

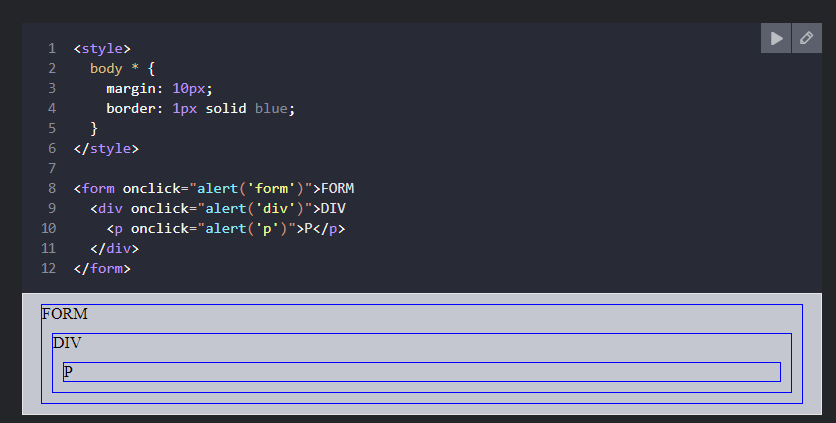
• obj.hasOwnProperty(“prop”) – проверяет наличие свойства только у obj, не идёт по цепочке прототипов

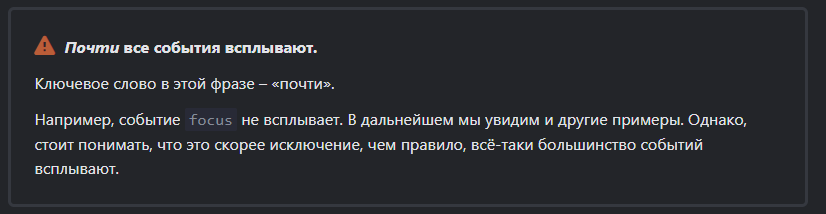
Проверка на присутствие в цепочке прототипов:

• instanceof – проверяет, принадлежит ли экземпляр (instance) к указанному классу (или к прототипу/прототипам класса)

1. **промисы, async/await**
2. **всплытие событий**

Клик на «P» вызовет обработчик событий у самого P и у каждой родительской ноды. Процесс называется **всплытием события.**



**!Исключение**  


**event.target**

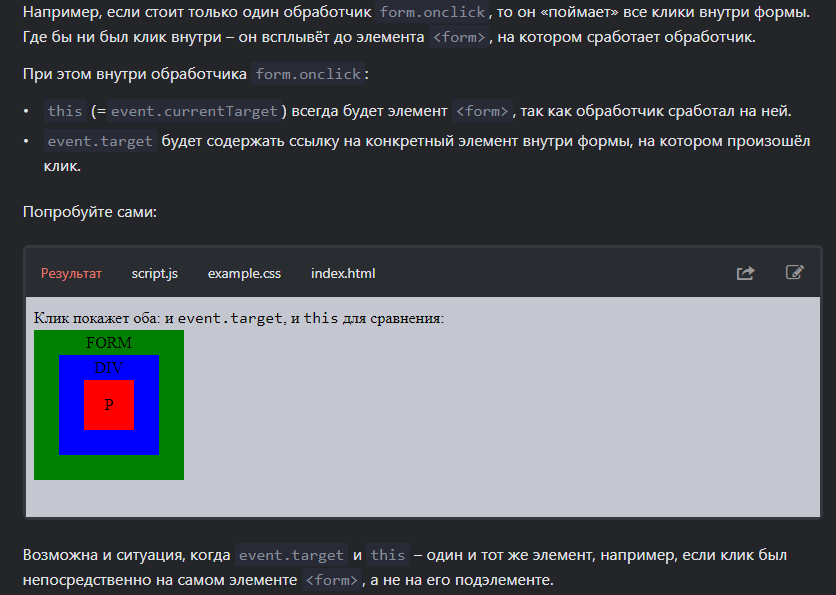
Самый глубокий элемент который вызывает всплытие называется *целевым элементом* и он доступен через event.target

Отличие от this:

• целевой элемент – элемент, с которого началось всплытие событий (т.е. на нем и произошло событие)

• this – текущий элемент, до которого дошло всплытие

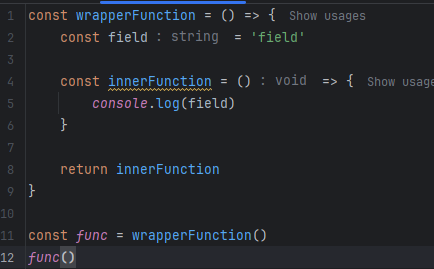
Пример:

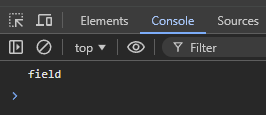


Прекратить всплытие события можно посредством event.stopPropagation()

1. **замыкание**

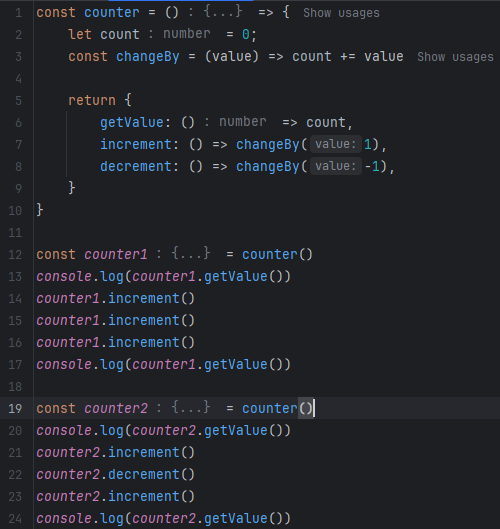
Замыкание – это комбинация функции и лексического окружения, в котором эта функция была определена.

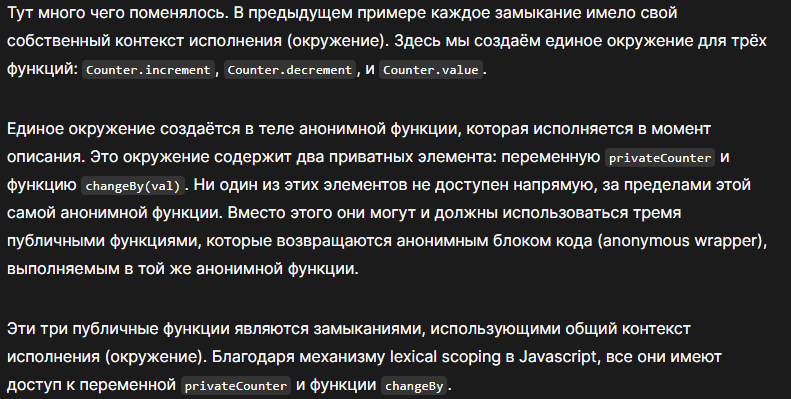




Область применения: эмуляция private полей/методов, организация глобального пространства имён.

Пример эмуляции private поля count и использование публичного геттера getValue для извлечения значения (напрямую без геттера работать не будет):





1. **call, apply, bind**
2. **garbage collector**
3. **== and ===**
4. **Spread & rest operators**
5. **Map and WeakMap, Set and WeakSet**
6. **Функции высшего порядка**