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

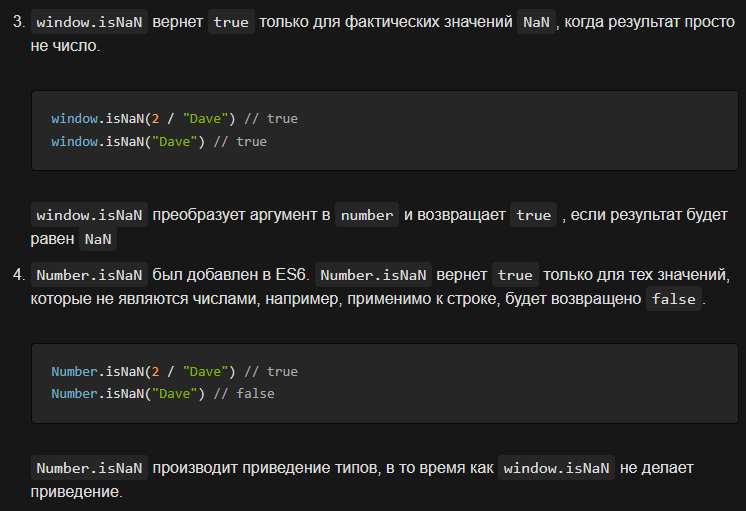
typeof NaN // “number”! // Не число, которое является числом

**typeof NaN**

1. Никогда не равен сам себе

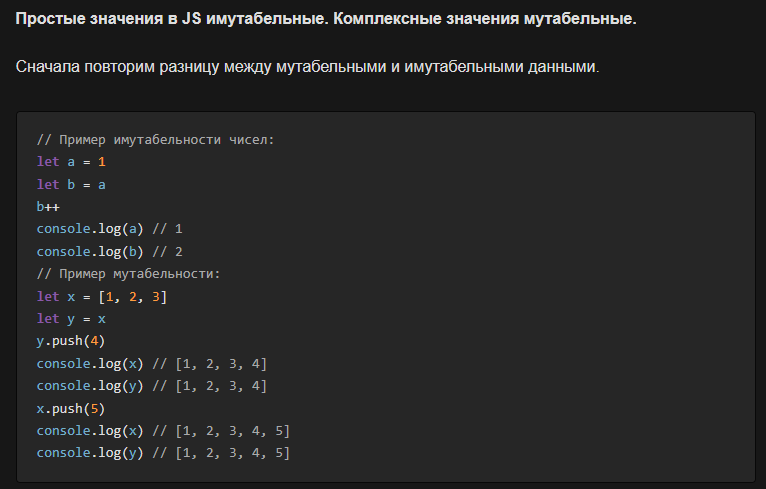
NaN === NaN //false

1. Проверка на NaN:



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

Пример мутабельности / имутабельности



1. **Преобразование и сравнение типов**
   1. **Примитивы**

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

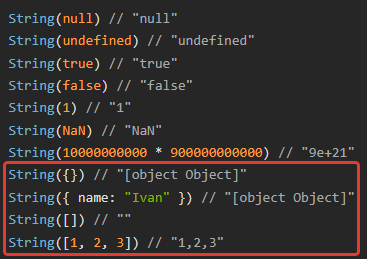
String(123) // ”123”

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

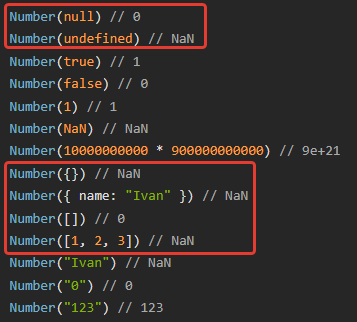
“2” + “2” = “22” //конкатенация строк

Преобразовать любой тип данных можно только к 3м типам:

• string:



• number:



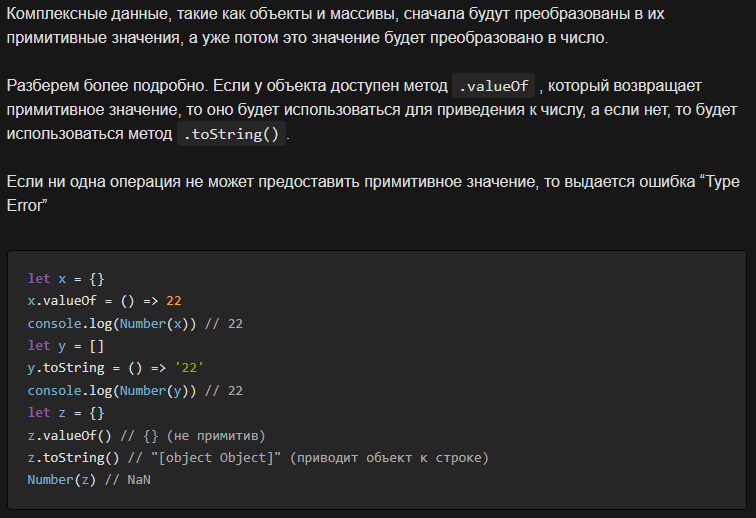
• bool

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

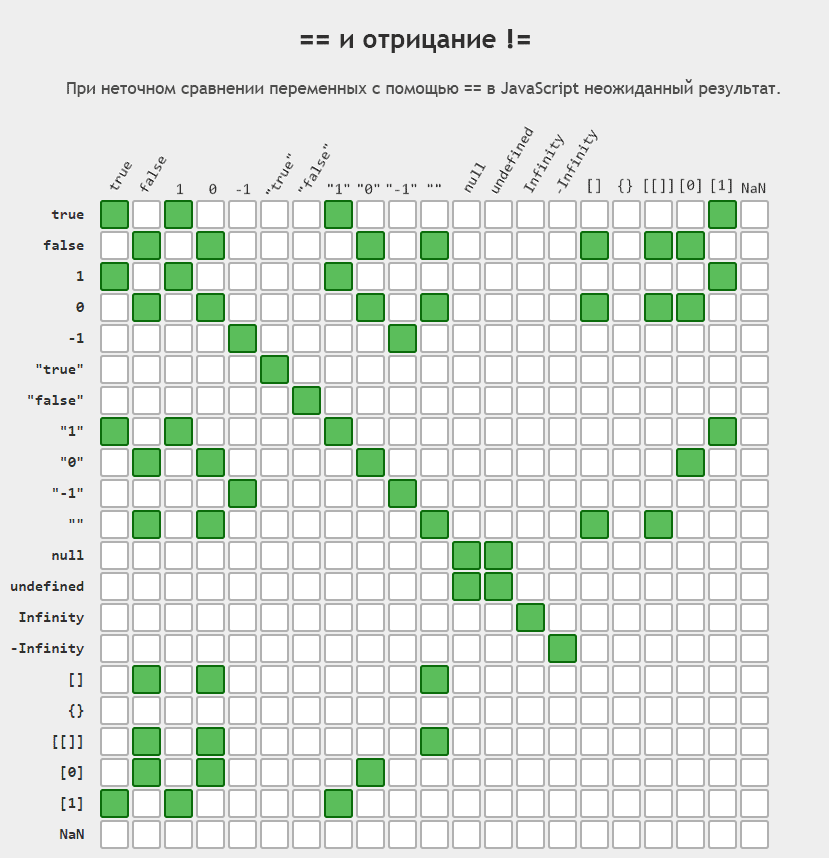
null, undefined, “”, 0, NaN; //falsy типы

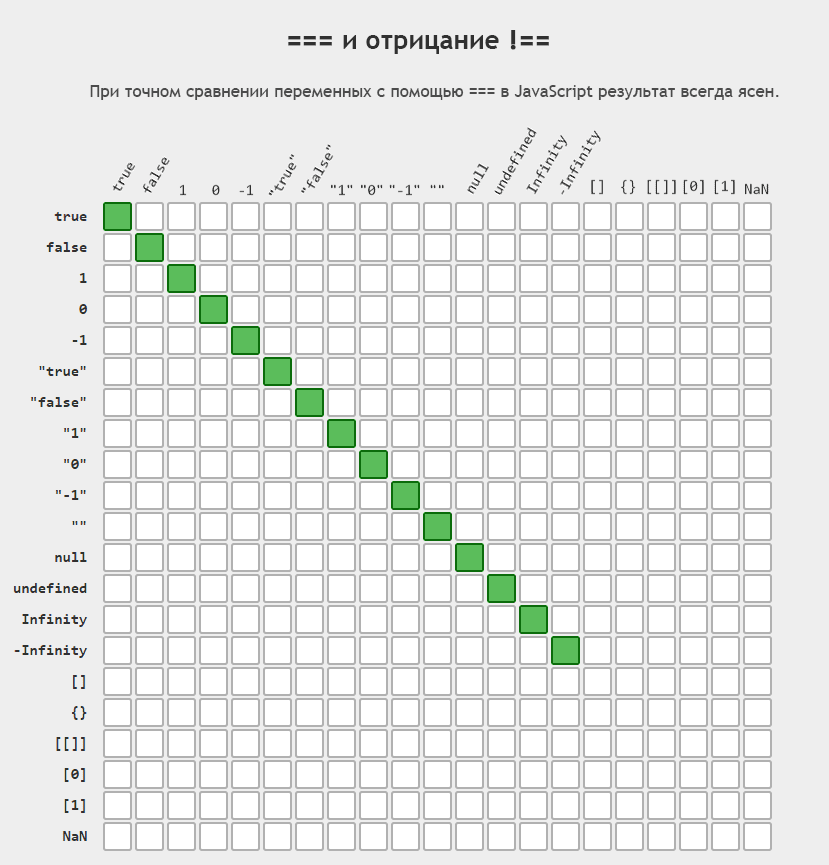
Неявное приведение между строкой и числом:

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



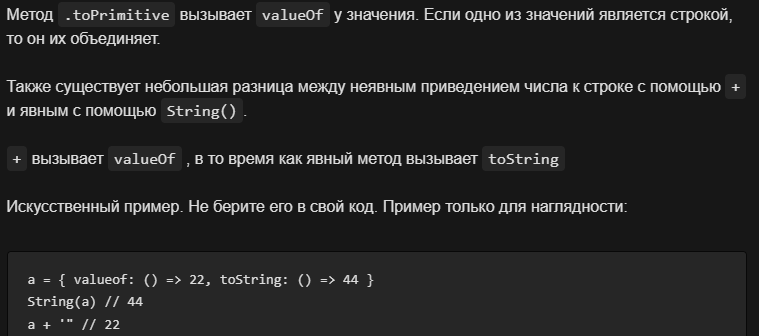
1. **Строгое сравнение (===) и сравнение с приведением типов (==)**



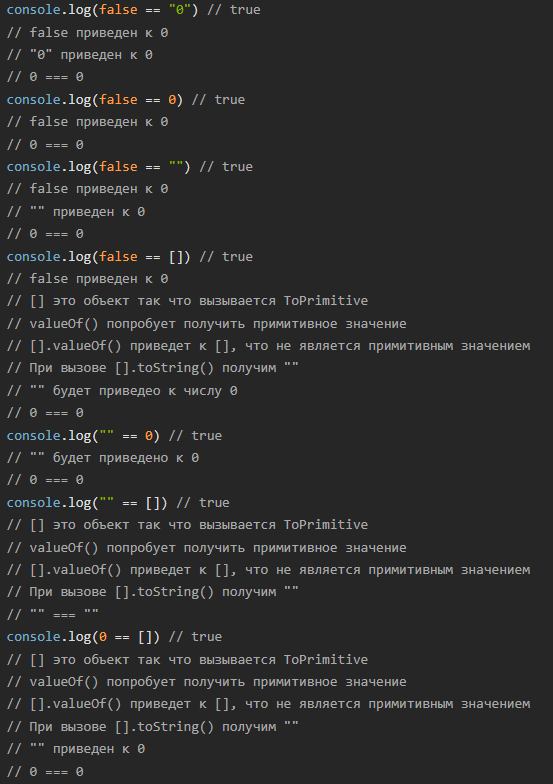


1. **Особенности**
2. Откуда у примитивов типа “string” есть методы вроде .toUpperCase?

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