



# ТИПЫ ДАННЫХ



МИХАИЛ КУЗНЕЦОВ



# МИХАИЛ КУЗНЕЦОВ

Разработчик в ING Bank



[@mkuznetcov](https://www.instagram.com/mkuznetcov)



# ПЛАН ЗАНЯТИЯ

1. [Типы данных](#)
2. [Примитивы и объекты. Данные по значению и по ссылке.](#)
3. [Обертки для примитивов](#)
4. [Методы типов данных](#)
5. [Проверка и преобразование типов](#)



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Какие типы данных существуют в JavaScript?

# ТИПЫ ДАННЫХ

```
1 let a = 5; // number
2 let b = "это строка"; // string
3 let c = true; // boolean
4 let d; // undefined
5 let e = null; // null
6 let f = { a:1, b:false }; // object
7 let g = function(){}; // function (о них будет следующая лекция)
```

Все ли это возможные типы?

# НОВЫЕ ТИПЫ

```
1 let s = new Symbol("symbol_name"); // Symbol
2 const theBiggestInt = 9007199254740991n; // BigInt
3 const alsoHuge = BigInt(9007199254740991); // BigInt
4 const hugeHex = BigInt("0x1fffffffffffffffff"); // BigInt
```

- [Symbol](#)
- [Symbol на learn.javascript.ru](#)
- [BigInt](#)
- [BigInt — новый тип данных в JS](#)

В реальной практике эти типы используются очень редко. Поэтому они выходят за рамки данного курса.



# ПРИМИТИВЫ И ОБЪЕКТЫ

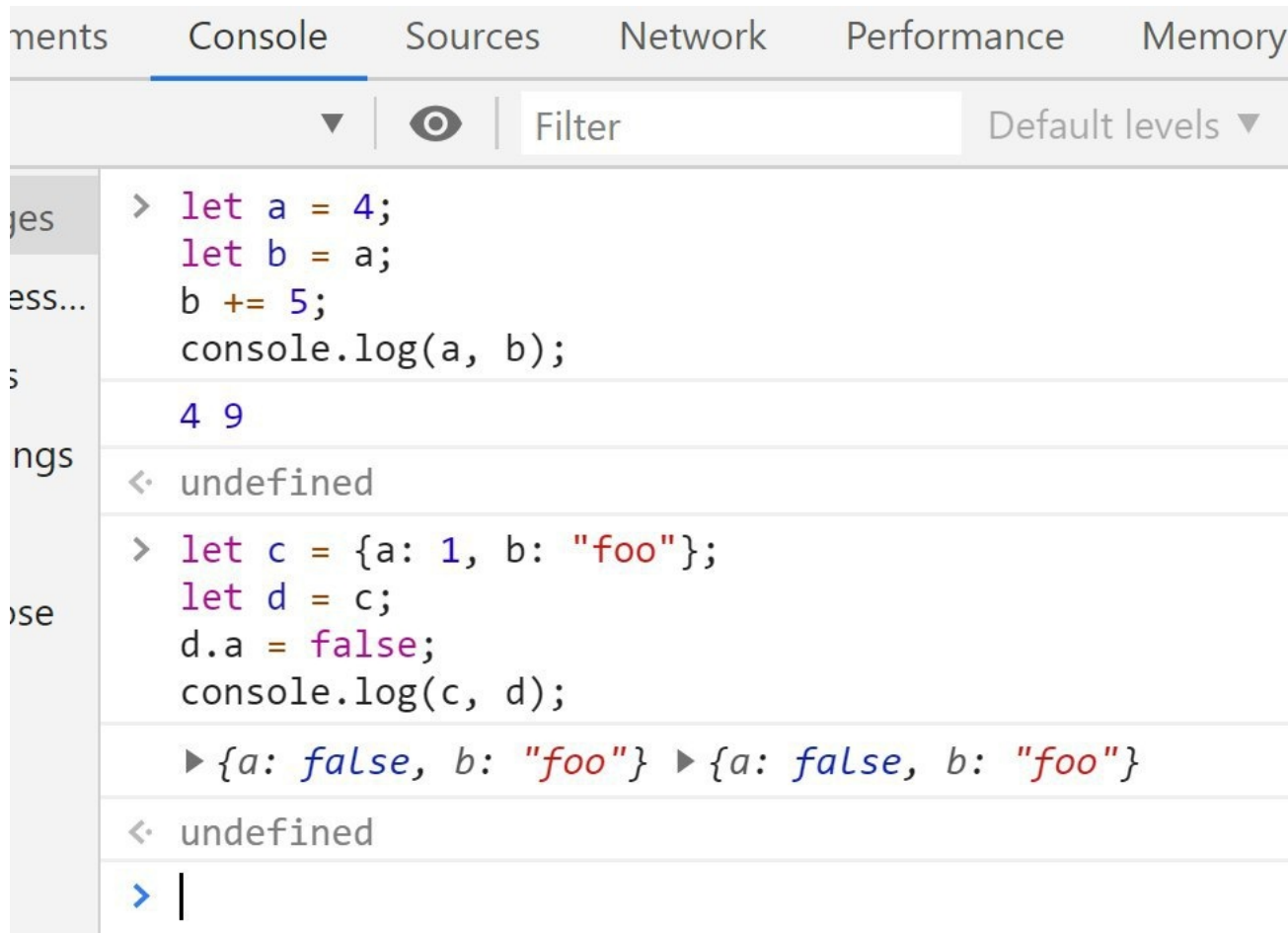
# ДАННЫЕ ПО ЗНАЧЕНИЮ И ПО ССЫЛКЕ (ДЕМО)

Вопрос, что будет выведено в результате выполнения кода?

```
1  let a = 4;  
2  let b = a;  
3  b += 5;  
4  console.log(a, b); // (1)  
5  // a так?  
6  let c = {a: 1, b: "foo"};  
7  let d = c;  
8  d.a = false;  
9  console.log(c, d); // (2)
```



# ДАННЫЕ ПО ЗНАЧЕНИЮ И ПО ССЫЛКЕ. РЕЗУЛЬТАТ

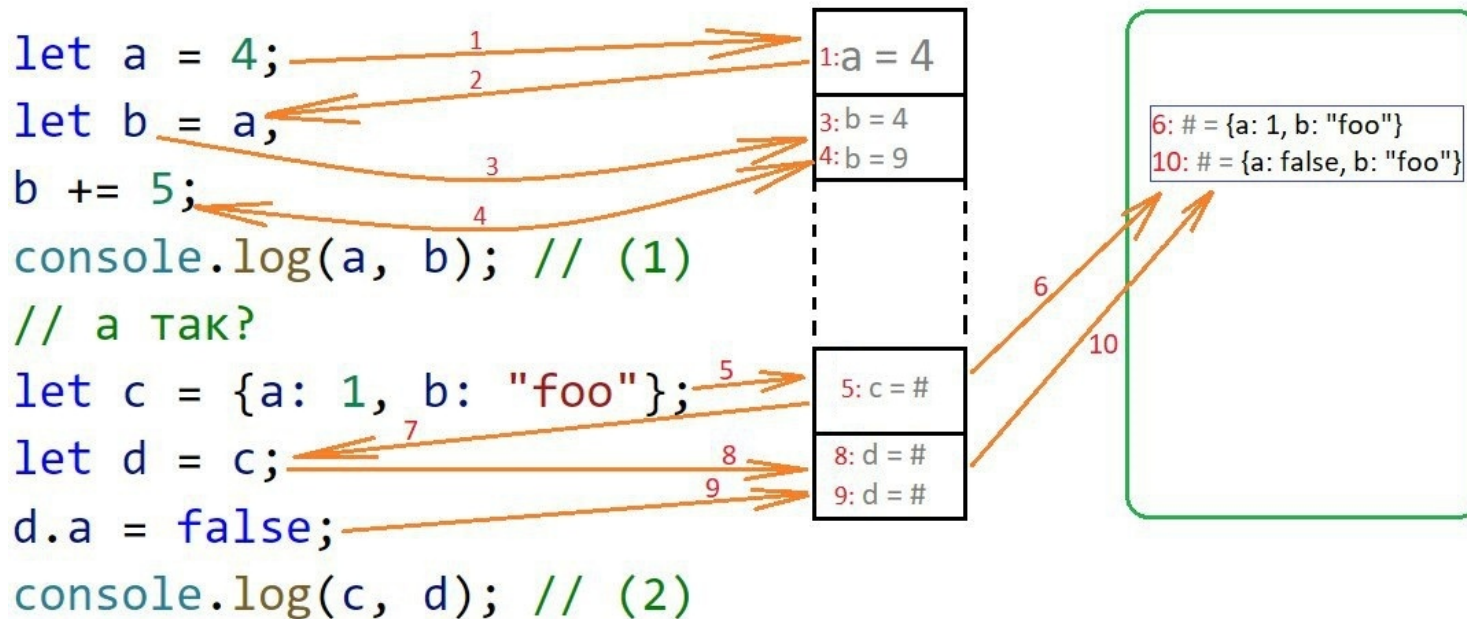


The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following code and its results:

```
> let a = 4;  
    let b = a;  
    b += 5;  
    console.log(a, b);  
  
4 9  
  
< undefined  
  
> let c = {a: 1, b: "foo"};  
    let d = c;  
    d.a = false;  
    console.log(c, d);  
  
▶ {a: false, b: "foo"} ▶ {a: false, b: "foo"}  
  
< undefined  
  
> |
```

The first code block defines variables `a` and `b`, increments `b` by 5, and logs the values of `a` and `b`. The result is `4 9`. The second code block defines an object `c` with properties `a` and `b`, creates a reference `d` to `c`, changes `d.a` to `false`, and logs both objects. The result shows two object literals: `{a: false, b: "foo"}` and `{a: false, b: "foo"}`. The console also shows `< undefined` for the return values of the log statements.

# ДАННЫЕ ПО ЗНАЧЕНИЮ И ПО ССЫЛКЕ. ПОЯСНЕНИЕ

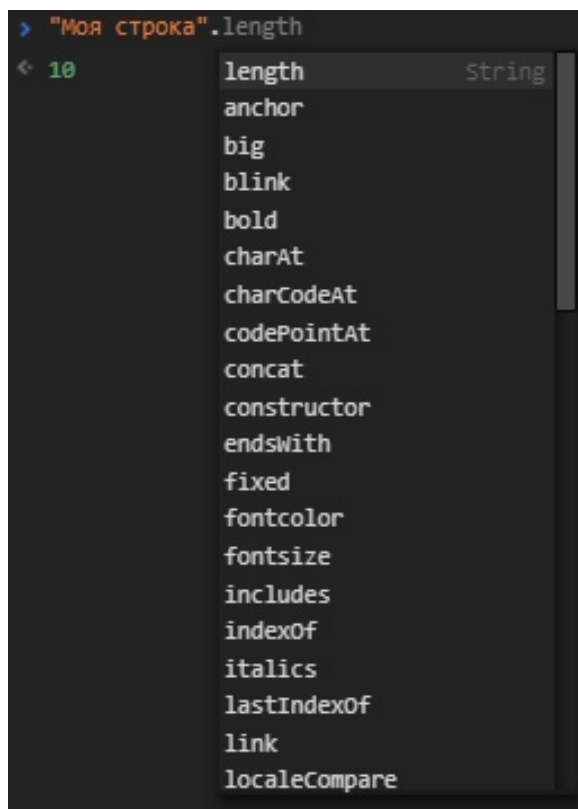




# ОБЕРТКИ ДЛЯ ПРИМИТИВОВ

# МЕТОДЫ ПРИМИТИВОВ

При создании примитива у некоторых по умолчанию есть встроенные методы:



```
> "Моя строка".length
10
length
anchor
big
blink
bold
charAt
charCodeAt
codePointAt
concat
constructor
endsWith
fixed
fontcolor
fontsize
includes
indexOf
italics
lastIndexOf
link
localeCompare
```

**Вопрос:** откуда взялись эти методы?

# МЕТОДЫ ПРИМИТИВОВ

**Ответ:** для примитива создается объект-обертка, у которого находятся все эти методы.

Пояснение: При получении значения `"Моя строка".length` браузер делает следующее:

1. Создаёт обёртку для используемого значения ("Моя строка" в данном случае).
2. Возвращает значение свойства (`length` в данном случае) или метода.
3. Удаляет ненужную обертку.

**Осторожно!** Везде где нужно, обертки создаются автоматически. Стоит избегать ручного их создания в коде.

# ОБЕРТКИ ДЛЯ ПРИМИТИВОВ

Как создать объект обертку над примитивом?

```
1 let num = new Number(1.123);  
2 let str = new String("Привет");  
3 console.log(num.toFixed(2)); // 1.12  
4 console.log(num.toUpperCase()); // ПРИВЕТ
```

# ОБЕРТКИ ДЛЯ `null` И `undefined`

**Вопрос:** для всех примитивов существуют обертки?

# ОБЕРТКИ ДЛЯ `null` И `undefined`

**Ответ:** нет, для `null` и `undefined` их не существует.

```
1 new Null(); // ReferenceError: Null is not defined
2 new null(); // TypeError: null is not a constructor
3 new Undefined(); // ReferenceError: Undefined is not defined
4 new undefined(); // TypeError: undefined is not a constructor
```

Для этих типов существует только одно значение и создавать целый объект для них не имеет смысла.



# ПОЛУЧЕНИЕ ЗНАЧЕНИЕ ИЗ ОБЪЕКТА ОБЕРТКИ

У каждого объекта обертки есть метод `valueOf()`, с помощью которого можно получить значение примитива.

```
let bool = new Boolean(true);  
console.log(bool.valueOf()); // true
```

# ОСОБЕННОСТИ ПРИ РАБОТЕ С ОБЪЕКТАМИ ОБЕРТКАМИ

При работе с объектами обертками необходимо соблюдать осторожность.  
Что будет выведено на консоль?

```
1 | let bool = new Boolean(false);  
2 | console.log(bool ? true : false);
```

## ПОЧЕМУ ВЫВЕЛОСЬ `true` ? (ДЕМО)

```
1 let simpleBool = false;
2 let wrapperBool = new Boolean(simpleBool); // создали объект обертку
3 // какого типа переменные?
4 console.log(typeof simpleBool); // boolean
5 console.log(typeof wrapperBool); // object т.к. это объект
6 // приведение любого объекта к логическому типу возвращает true
```



# МЕТОДЫ ТИПОВ ДАННЫХ

# ЧИСЛА

Все числа в JavaScript, как целые так и дробные, имеют тип Number и хранятся в 64-битном формате [IEEE-754](#).

```
1 let a = 54; // 54 в десятичной системе
2 let b = 0xFF; // 255 в шестнадцатеричной системе
3 let c = 3e5; // в научной форме: 3 с 5 нулями т.е. 300000
4 let d = 3e-5; // здесь 3 сдвинуто 5 раз вправо, за десятичную точку т.е. 0.00003
```

# НЕТОЧНЫЕ ВЫЧИСЛЕНИЯ

Какое значение переменной `result` после вычисления:

```
let result = (0.1 + 0.2 == 0.3);
```

`true` или `false`?

# НЕТОЧНЫЕ ВЫЧИСЛЕНИЯ

Операции над числами могут привести к неожиданным результатам:

```
alert(0.1 + 0.2); // 0.30000000000000004  
alert(9999999999999999); // 10000000000000000
```

Всё дело в том, что в стандарте IEEE 754 на число выделяется ровно 8 байт (= 64 бита), не больше и не меньше.

# КАК ИЗБЕЖАТЬ НЕТОЧНЫХ ВЫЧИСЛЕНИЙ?

Функция `toFixed(digits)` форматирует число, используя запись с фиксированной запятой.

`digits` - количество цифр после десятичной запятой; может быть значением между 0 и 20 включительно, хотя реализации могут поддерживать и больший диапазон значений. Если аргумент опущен, он считается равным 0.



# ОКРУГЛЕНИЕ ЧИСЕЛ

```
1 Math.floor() // Округляет в меньшую сторону
2 Math.round() // Округляет к ближайшему целому
3 Math.ceil() // Округляет в большую сторону
4 Math.trunc() // Убирает дробную часть
```

# ОКРУГЛЕНИЕ ЧИСЕЛ

Что будет выведено на консоль?

```
1 console.log(Math.floor(1.1));  
2 console.log(Math.floor(1.5));  
3 console.log(Math.floor(1.9));  
4  
5 console.log(Math.round(2.1));  
6 console.log(Math.round(2.5));  
7 console.log(Math.round(2.9));  
8  
9 console.log(Math.ceil(3.1));  
10 console.log(Math.ceil(3.5));  
11 console.log(Math.ceil(3.9));
```

# ЗНАЧЕНИЕ Infinity

**Вопрос:** какое значение мы получим, если попытаемся какое-либо число разделить на ноль?

## ЗНАЧЕНИЕ `Infinity`

```
console.log(1/0); // Infinity
```

Это значение ведёт себя как математическая бесконечность; например, любое положительное число, умноженное на `Infinity` даёт `Infinity`, а любое число, делённое на `Infinity`, даёт `0`.

## ОСОБЕННЫЕ ЧИСЛОВЫЕ ЗНАЧЕНИЯ №2

Какое значение мы получим, если попытаемся строку умножить или разделить на число?

Если мы попытаемся получить число из значения, которое ввели в поле для ввода, то не всегда у нас может получиться ожидаемое значение.

NaN является значением, представляющим не-число (Not-A-Number).

```
console.log("десять" * 3); // NaN
```

## ЗНАЧЕНИЕ NaN

Возникает, когда математические функции не могут вернуть значение (например, при вызове `Math.sqrt(-1)`) или когда функция считывания числа из строки не может это сделать, потому что в строке не число (`parseInt('blabla')`).

```
1 console.log(typeof NaN); // "number"
2 console.log(NaN === NaN); // false
3 console.log(isNaN(NaN)); // true
```

# СТРОКИ

В JavaScript любые текстовые данные являются строками.

Строки создаются при помощи двойных или одинарных кавычек.

```
1  var text = "это строка";  
2  var anotherText = 'еще одна строка';  
3  var str = "012345";
```

# ШАБЛОННЫЕ СТРОКИ

В JavaScript строки можно составлять по шаблону вставляя определенные значения непосредственно в строку, для этого необходимо воспользоваться косыми кавычками:

```
`это шаблонная строка`
```

```
let name = "Петр";  
console.log(`Привет, меня зовут ${name}, мне ${10+9} лет`);
```



# СПЕЦИАЛЬНЫЕ СИМВОЛЫ

Как решить проблему, если необходимо вставить символ, который отсутствует на клавиатуре или необходимо добавить кавычки?

Символ	Описание
--------	----------

<code>\n</code>	Перевод на новую строку
-----------------	-------------------------

<code>\t</code>	Символ табуляции
-----------------	------------------

<code>\uNNNN</code>	Любой юникод символ с шестнадцатеричным кодом
---------------------	---

<code>\'</code>	Экранирование одинарной кавычки
-----------------	---------------------------------

<code>\"</code>	Экранирование двойной кавычки
-----------------	-------------------------------

# СПЕЦИАЛЬНЫЕ СИМВОЛЫ

```
1 console.log('I\'m a JavaScript programmer'); // I'm a JavaScript programmer
2 console.log('\u262D \u262A \u2766 \u2713 \u262F \u2328'); // 🤔 🌟 🌀 ✓ 🙄 📺
```

# ДОСТУП К СИМВОЛАМ

Строка является массивом символов, следовательно можно получить символ или воспользоваться методом `charAt()`.

```
1 | let myString = "моя строка";  
2 | console.log(myString[4]); // 'с'  
3 | console.log(myString.charAt(4)); // 'с'
```

## РАЗЛИЧИЯ МЕЖДУ ОБРАЩЕНИЕМ КАК К МАССИВУ И ФУНКЦИЕЙ `charAt()`

```
1 | let myString = "моя строка";  
2 | console.log(myString[30]); // undefined  
3 | console.log(myString.charAt(30)); // ""
```

# ДЛИНА СТРОКИ

Одно из самых частых действий со строкой – это получение ее длины. Длина строки находится в свойстве `length`.

Если нам необходимо отправить значение на сервер, для сохранения в базе данных, стоит задуматься о валидации этого значения, что бы оно не было слишком длинным.

```
let myString = "моя строка";  
console.log(myString.length); // 10
```

# СМЕНА РЕГИСТРА

Методы `toLowerCase()` и `toUpperCase()` меняют регистр строки на нижний/верхний.

Например если необходимо сравнить ввод пользователя с ключевой фразой без учета регистра.

```
console.log("моя строка".toUpperCase()); // "МОЯ СТРОКА"  
console.log("Моя СтРоКа".toLowerCase()); // "моя строка"
```

# ПОИСК СТРОКИ В СТРОКЕ

Для поиска строки в строке есть метод

```
indexOf(искомая_строка[, начальная_позиция]).
```

Он возвращает позицию, на которой находится подстрока или -1, если ничего не найдено. Например:

```
console.log("моя строка".indexOf("о")); // 1  
console.log("Моя СтРоКа".indexOf("о", 4)); // 7
```

# ВЗЯТИЕ ПОДСТРОКИ

В JavaScript существуют целых 2 метода для взятия подстроки, с небольшими отличиями между ними.

1. `substring(start[, end])`;
2. `slice(start [, end])`.



# СРАВНЕНИЕ СИМВОЛОВ

Символы сравниваются в алфавитном порядке 'А' < 'Б' < 'В' < ... < 'Я', но с некоторыми особенностями.

Все строки имеют внутреннюю кодировку Юникод. По этому коду сравниваются строки.

```
console.log('а' > 'Я'); // true  
console.log('ё' > 'я'); // true, т.к. ё находится после строчных
```

# СРАВНЕНИЕ СИМВОЛОВ

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0400	È	Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ	Й	Ў	Ц
0410	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
0420	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
0430	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
0440	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
0450	è	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	й	ў	ц
0460	Ѓ	ѡ	Ђ	Ѣ	Ј	Ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ

# СРАВНЕНИЕ СТРОК

Сравнение строк работает лексикографически, т.е. посимвольно. Например, при вводе капчи необходимо проверить пользовательский ввод с правильным результатом.

Сравнение строк `str1` и `str2` обрабатывается по следующему алгоритму:

1. Сравняются первые символы: `str1[0]` и `str2[0]`. Если они разные, то сравниваем их и, в зависимости от результата их сравнения, вернуть `true` или `false`. Если же они одинаковые, то...
2. Сравняются вторые символы `str1[1]` и `str2[1]`.
3. Затем третьи `str1[2]` и `str2[2]` и так далее, пока символы не будут наконец разными, и тогда какой символ больше – та строка и больше.
4. Если же в какой-либо строке закончились символы, то считаем, что она меньше, а если закончились в обеих – они равны.

# ПОЛУЧЕНИЕ СИМВОЛА ПО КОДУ И КОД ПО СИМВОЛУ

Метод `String.fromCharCode(code)` возвращает символ по коду:

```
console.log(String.fromCharCode(8381)); // ₽
```

Метод `str.charCodeAt(pos)` возвращает код символа на позиции `pos`:

```
console.log("строка".charCodeAt(0)); // 1089, код 'с'
```

# ЛОГИЧЕСКИЙ ТИП `boolean`

Вспоминаем прошлые занятия.

- Как работают условные операторы?
- Как заканчиваются циклы?

# ЛОГИЧЕСКИЙ ТИП `boolean`

Тип `boolean` имеет только 2 значения `true` (истина) и `false` (ложь).

```
1 | let a = true;  
2 | let b = false;  
3 | let c = 5 > 3; // true
```

## ТИП `undefined`

Вспоминаем прошлые занятия.

- Чему равно значение по умолчанию объявленной переменной?

## КОГДА ИСПОЛЬЗУЕТСЯ `undefined`

Переменная, не имеющая присвоенного значения, обладает типом `undefined`. Также возвращают `undefined` метод или инструкция, если переменная, участвующая в вычислениях, не имеет присвоенного значения. Функция возвращает `undefined`, если она не возвращает какого-либо значения.

```
let a; // undefined
let b = console.log(2+2); // выведет 4
console.log(b); // выведет undefined
```

В явном виде `undefined` **никогда** не присваивают, так как это противоречит его смыслу. Для записи в переменную «пустого» или «неизвестного» значения используется `null`.



# ЛОГИЧЕСКИЕ СРАВНЕНИЯ **undefined**

```
// переменная x не была определена ранее  
typeof x === 'undefined' // вычислится в true без ошибок  
x === undefined // выкинет ReferenceError
```

```
let x;  
x === undefined // true
```

## ТИП `null`

Вспоминаем прошлые занятия.

- Какой тип мы используем, если хотим показать отсутствие значения?

## ТИП `null`

Переменная имеет значение `null`, если в ней нет явного значения.

Например если в списке автомобилей необходимо указать объем бака, то у электромобилей бак отсутствует

```
let a = null;
```

## СРАВНЕНИЕ `null` С НУЛЁМ

```
null > 0 // false
null == 0 // false
null >= 0 // true
```

Дело в том, что алгоритмы проверки равенства `==` и сравнения `>=` `>` `<` `<=` работают по-разному.

Сравнение честно приводит к числу, получается ноль. А при проверке равенства значения `null` и `undefined` обрабатываются особым образом: они равны друг другу, но не равны чему-то ещё.



## ВЫВОД (ДЕМО)

Следует избегать сравнений на больше-меньше с не-числовыми типами.

## ОТЛИЧИЯ МЕЖДУ `null` И `undefined`

```
1  typeof null    // object (баг в ECMAScript, должно быть null)
2  typeof undefined // undefined
3  null === undefined // false
4  null == undefined // true
```

# ПРЕОБРАЗОВАНИЯ ТИПОВ

Предположим, что у вас есть строка в которой хранится число, к которому необходимо прибавить число. Как это сделать?

```
// Что будет выведено?  
console.log("13" + 6);
```

## ПОЧЕМУ ПОЛУЧИЛОСЬ 136, А НЕ 19?

При сложении строк движок JavaScript преобразует все параметры к строке и складывает их.

```
console.log("13" + 6); // "13" + "6" = "136"  
console.log("мой "+" дом"); // "мой дом"
```





# РЕШЕНИЕ ПРОБЛЕМЫ

Вспоминаем прошлые занятия.

- Что делают функции `parseFloat()` и `parseInt()`?

# ЯВНЫЕ ПРЕОБРАЗОВАНИЯ ТИПОВ

Значение можно явно преобразовать из одного типа в другой

```
1 console.log(Number("13") + 6); // 19
2 console.log(String(123)); // "123"
3 console.log(Boolean(123)); // true
4 console.log(Object(123)); // Number {123} - мы получили объект
```

# НЕЯВНЫЕ ПРЕОБРАЗОВАНИЯ ТИПОВ

В некоторых случаях значения автоматически преобразуются.

```
1 // Вспоминаем прошлый пример
2 console.log("13" + 6); // "13" + "6" = "136"
3 // 6 автоматически преобразовалось к строке
4 console.log({ a : 1, b : "c" }) // { a:1, b : "c" }
5 // объект преобразовался к строке, чтобы вывестись в консоль
```

# ПРИНУДИТЕЛЬНЫЕ НЕЯВНЫЕ ПРЕОБРАЗОВАНИЯ

В некоторых случаях самостоятельно неявно преобразовать тип:

```
1 // с помощью 'Number' значения преобразуются к строке
2 console.log(7 + Number("25")); // 32, 25 преобразовалось к числу
3
4 // с помощью 'Boolean' или '!!' значения преобразуются к логическому типу
5 console.log(Boolean(1)); // true
6 console.log(Boolean("")); // false, т.к. строка пустая
7
8 console.log(!!0); // false
9 console.log(!!"0"); // true, т.к. строка не пустая
```

# ПРОБЛЕМЫ С ПРЕОБРАЗОВАНИЕМ К ЧИСЛУ

```
console.log(Number("25abc") + 7); // NaN, упс... Вызов Number не приемлет нечисловые строки  
console.log(parseInt("25abc") + 7); // 32, сработало
```



## ЧЕМУ МЫ НАУЧИЛИСЬ?

1. Подробно изучили примитивы;
2. Узнали про объекты-обертки над примитивами;
3. Узнали, откуда берутся встроенные методы примитивов;
4. Разобрались с явными и неявными преобразованиями типов.

# ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаем в чате Slack!
- Задачи можно сдавать по частям.
- Зачет по домашней работе проставляется после того, как приняты **все задачи**.



Спасибо за внимание! Время задавать вопросы

**МИХАИЛ КУЗНЕЦОВ**

