Лекция 3. Java Script

Интернет-технологии, 3 курс

О языке

- С языком Java не имеет ничего общего. Название было дано из-за ажиотажа на Java в момент выпуска.
- На текущий момент используется как для программирования клиентской логики (браузеры), так и для серверной логики (nodejs).
- Популярность получил благодаря гибкости и одному из наименее избыточных форматов передачи данных JSON.
- Является одним из наиболее спорных языков из за своей архитектуры.
- Является языком с автоматической сборкой мусора.

Прототипное наследование

- Основная идея заключается в том, чтобы отказаться от чистых типов (классов, структур) как таковых.
- Каждый объект в JavaScript сам описывает себя полностью. Все методы объекта включаются непосредственно в сам объект.
- При наследовании в JavaScript объекты наследуют свойства от других объектов. Меняя базовый объект, вы меняете его наследника.
- Из-за потенциальной избыточности такого подхода существуют способы создания функций-конструкторов, которые позволяют создавать типовые объекты.

Типизация и основные проблемы

- JavaScript – язык с динамической типизацией.
- Возможны только ручные проверки типов с помощью оператора **typeof** и оператора **instanceof**.
- Все поля объектов публичны, стандартная инкапсуляция не доступна.
- При создании и обработке больших структур анонимные объекты способны значительно занять память.

Типы данных

Базовые

- Число (Number)
- Строк (String)
- Логическое (Boolean)

Составные

- Функция (Function)
- Объект (Object)
- Массив (Array)

Специальные

- Null
- Undefined

1. **Number** включает в себя все стандартные числовые типы данных, свойственные **C**-подобным языкам.
2. **Функции**, по сути, являются разновидностью объектов, но при этом вынесены в отдельный тип при определении.
3. **Массивы** относятся к типу объекта. Чаще всего в отдельный тип выделяются из-за нативной реализации средствами стандарта **ECMAScript**

Числа

- JavaScript отличается от таких языков программирования, как C и Java, тем, что не делает различия между целыми и вещественными значениями.
- Все числа в JavaScript представляются 64 разрядными вещественными значениями (с плавающей точкой), формат которых определяется стандартом IEEE 754.

- Число, находящееся непосредственно в коде JavaScript программы, называется числовым литералом. JavaScript поддерживает числовые литералы следующих форматов:
  - Целые литералы
  - Шестнадцатеричные и восьмеричные литералы
  - Литералы вещественных чисел
  - Специальные числовые значения

Числовые литералы

Целые литералы

- 0
- 7
- 10293

Литералы вещественных чисел

- 3.14
- 2345.789
- .33333333333333333
- 6.02e23
- 1.4738223E32

Специальные числовые значения

- NaN – нечисловое значение.
- Infinity – бесконечность (может быть отрицательным)

Шестнадцатеричные и восьмеричные литералы

Шестнадцатеричные начинаются с 0x и 0X.
Восьмеричные 0

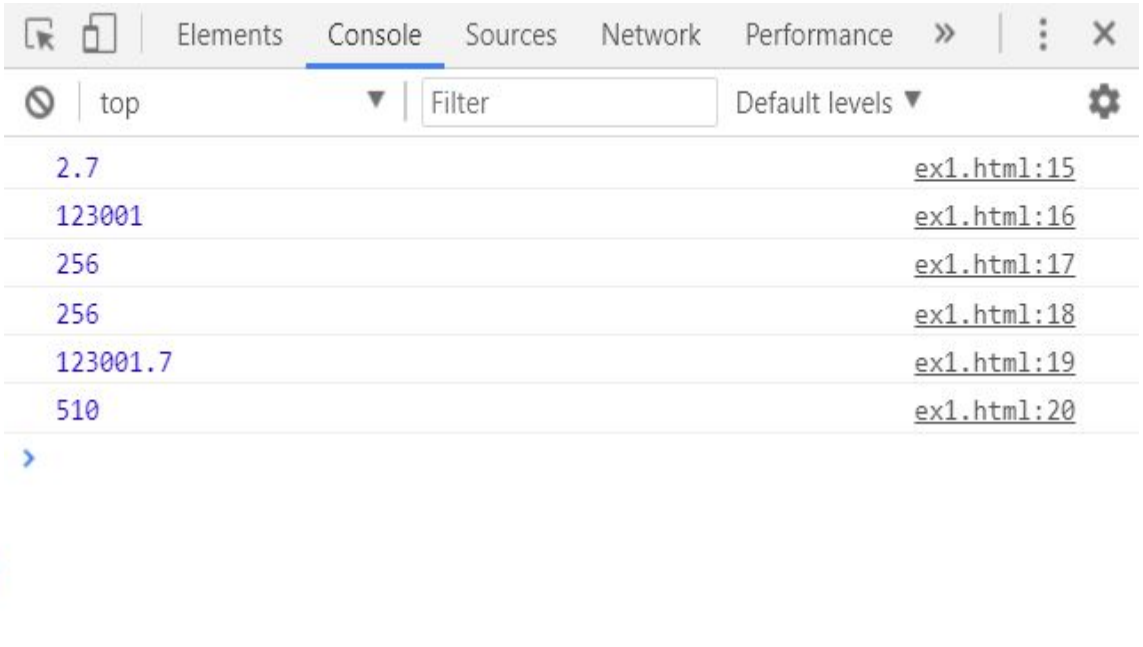
- 0xfa3
- 0X3d73
- 0377
- 0254

Операции с числами

+, -, *, /

Пример математических операций между разными литералами

```
var a = 1,
    b = 1.7,
    c = 1.23e5,
    d = 0377,
    e = 0xff;

console.log(a + b); // 2.7
console.log(a + c); // 123001
console.log(a + d); // 256
console.log(a + e); // 256
console.log(b + c); // 123001.7
console.log(d + e); // 510
```



Строки

- В JavaScript нет символьного типа данных, такого как char в C, C++ и Java. Одиночный символ представлен строкой единичной длины.
- Строковый литерал – это последовательность из нуля или более Unicode символов, заключенная в одинарные или двойные кавычки.
- Сами символы двойных кавычек могут содержаться в строках, ограниченных символами одинарных кавычек, а символы одинарных кавычек – в строках, ограниченных символами двойных кавычек.

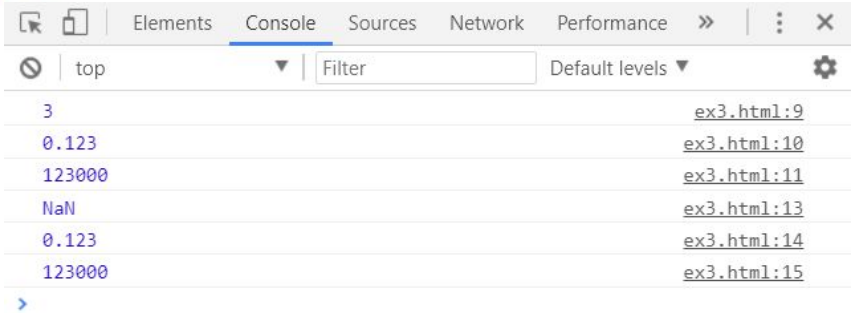
Преобразования строк в числа

Чаще всего преобразованию подвергаются данные получаемые извне:

- Методы ввода;
- AJAX запросы.

Преобразование числа в строку возможно осуществить с помощью встроенных функций parseInt и parseFloat, а также с помощью оператора унарного +. Работают они примерно одинаково, за исключением того, что унарный + не отсекает буквенные значения из строки к которой применяется, вместо чего дает значение NaN

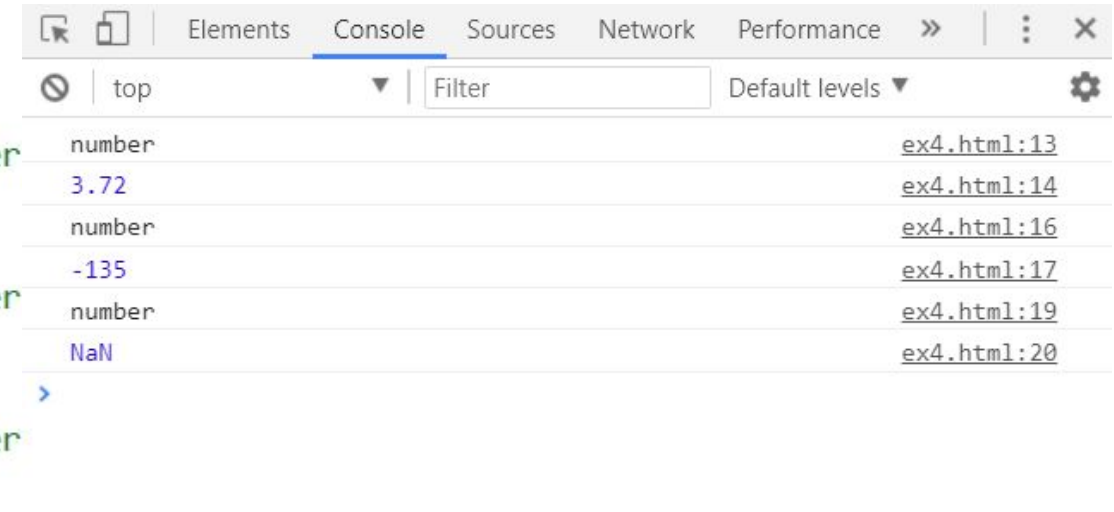
```
console.log(parseInt("3.267 не число ПИ")); // 3
console.log(parseFloat(".123")); // 0.123
console.log(parseFloat("1.23e5")); // 123000

console.log(+"3.267 не число ПИ"); // NaN
console.log(+".123"); // 0.123
console.log(+"1.23e5"); // 123000
```



Числовые строки

- Числовой строкой называется строка, содержащая один из числовых литералов.
- Данный вид строк может использоваться как операнд в математических и логических выражения без явного приведения. Исключением является операция +.
- Если числовая строка содержит недопустимые символы, то результатом ее преобразования будет являться NaN.

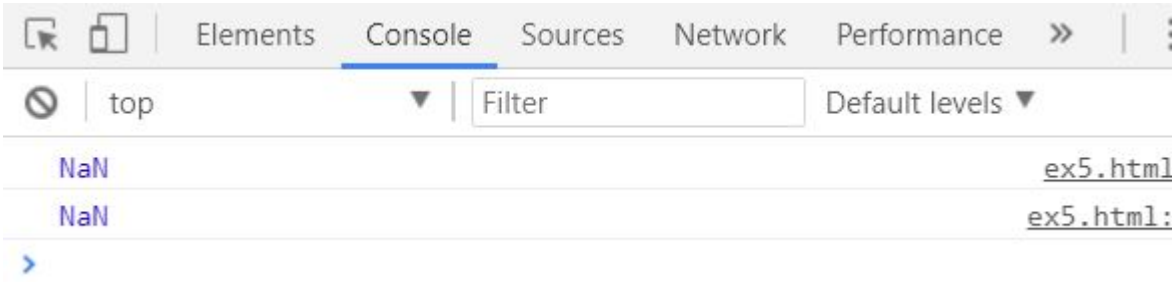
```
var res1 = "12.4" * "2.1" / "7",
    res2 = "1.2e2" - "0xff",
    res3 = "234h" - "17";

console.log(typeof res1); // number
console.log(res1); // 3.72

console.log(typeof res2); // number
console.log(res2); // -135

console.log(typeof res3); // number
console.log(res3); // NaN
```



Нечисловое значение NaN

- Данный специальный литерал получается в результате не возможности расчета при выполнения тех или иных математических действий.
- Основной причиной является использование в математическом выражении строки, не числовой строки.
- Если одним из операндов является NaN то результат всего математического выражения всегда будет NaN.

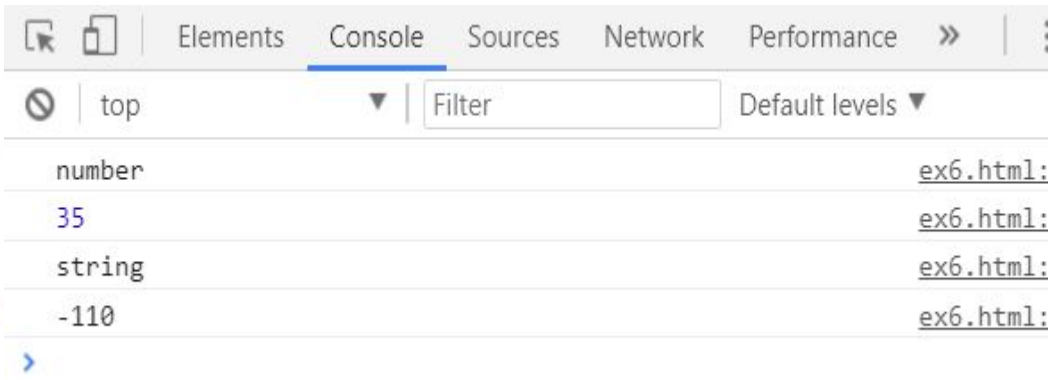
```
console.log((10 / 4.3) * "37d"); // NaN
console.log(NaN * 59) // NaN
```



Математические выражения с операцией «+»

- В JavaScript оператор **+** используется как для сложения чисел, так и для сложения строк. При этом результат зависит только от того, какие типы операндов присутствуют в выражении.
- Операция сложения строк имеет больший приоритет. Если один из операндов при сложении является строкой, то результатом будет тоже строка.
- При этом, результирующий тип всего составного выражения зависит от того, какая операция была выполнена последней.

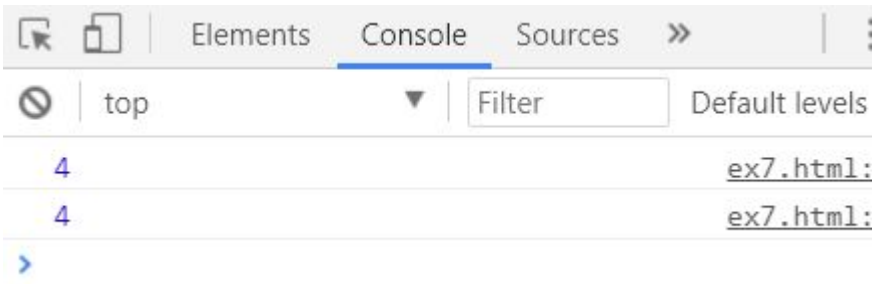
```
var res1 = "4" + 4 - 9,
    res2 = ("3" - 5 ) / 2 + "10";

console.log(typeof res1); // number
console.log(res1) // 2.54

console.log(typeof res2); // string
console.log(res2) // -110
```



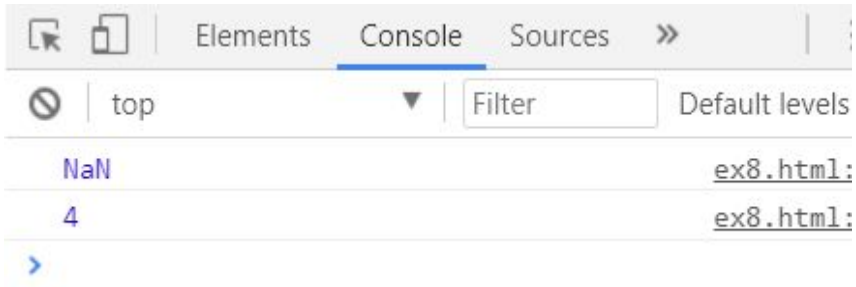
Решение проблемы «+»

- Всегда в своих выражениях преобразуйте в числа те переменные, которые могут является строкой.
- Если строки могу содержать символы не допустимые для чисел используйте функции.
- Если вы уверены в формате числовых строк, то достаточно будет унарного +. Но учтите, в случае ошибки в числовой строке, результатом выражения будет NaN.
- Выбор того или иного способа зависит от решаемой задачи. Что должно происходить при ошибке и должны ли игнорироваться неправильные операнды.

```
var a = "7";

console.log(17 + +a - 20); // 4
console.log(17 + parseInt(a) - 20); // 4
```



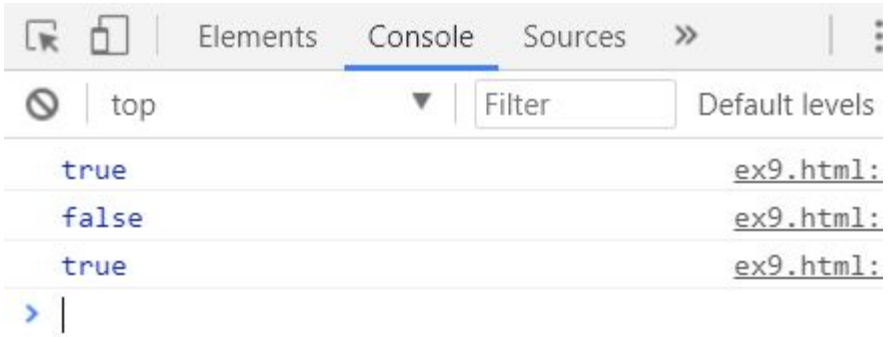

```
var a = "7b";

console.log(17 + +a - 20); // NaN
console.log(17 + parseInt(a) - 20); // 4
```



Логические

- Логический тип данных имеет только два допустимых значения, представленных литералами **true** и **false**.
- Логические значения обычно представляют собой результат сравнений, выполняемых в JavaScript.
- В случае наличия числового операнда, при сравнениях оба операнда приводятся к числовому значению.

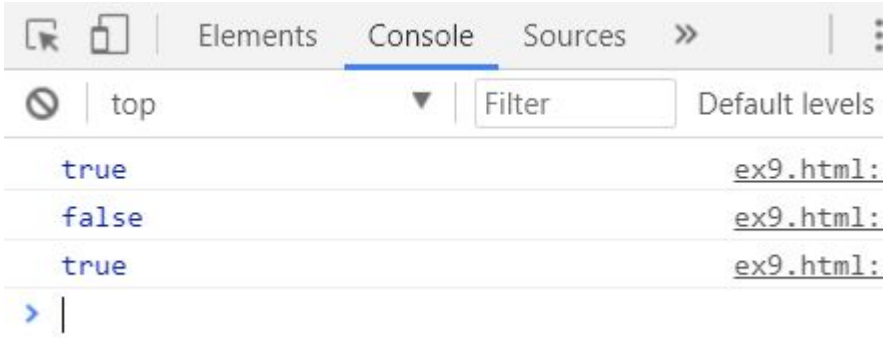
```
var b1 = 201 > "21",
    b2 = "201" > "21",
    b3 = 201 > 21;

console.log(b1); // true
console.log(b2); // false
console.log(b3); // true
```



Сравнение строк и чисел

- Механизмы сравнения строк и чисел идентичны всем С подобным языкам.
- В случае сравнения чисел, большим является то число, значение которого больше.
- В случае сравнения строк большей является та строка, у которой первый символ при расхождении больше.
- На предыдущем слайде при сравнении строк «201» и «21» расхождение было на втором символе. У строки «21» второй символ больше, поэтому и сама строка считается большей.

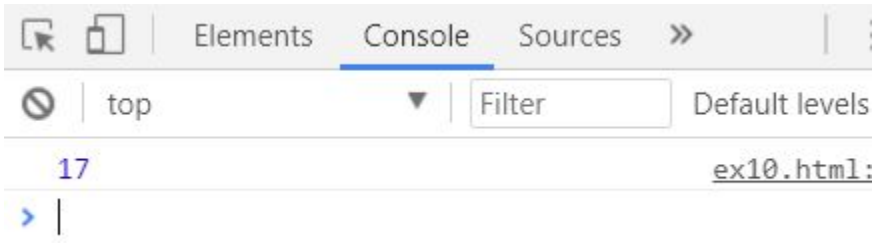
```
var b1 = 201 > "21",
    b2 = "201" > "21",
    b3 = 201 > 21;

console.log(b1); // true
console.log(b2); // false
console.log(b3); // true
```



Функции

- Функция – это фрагмент исполняемого кода, который определяется под некоторым именем, имеет свою область видимости и может быть вызван многократно.
- Функции могут иметь аргументы.

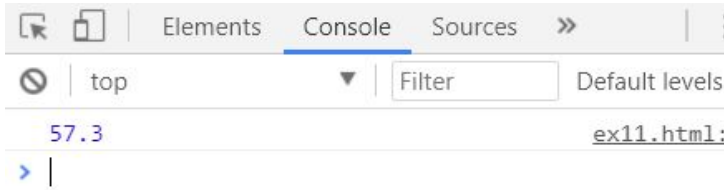
```
function testFunc(a){
    console.log(a);
};

testFunc(17);
```



Объекты

- В языке JavaScript отсутствуют типы. Возможнс только создание анонимных объектов или объектов формируемых с помощью функции-конструктора.
- Создание анонимных объектов происходит с помощью фигурных скобок. Внутри фигурных скобок указываются поля объекта. Имя поля и его значение разделены двоеточием. Между собой поля разделяются запятой.
- Полем может быть как объект базового типа, так и другой объект или функция.
- В случае добавления метода в объект внутри него под ключевом словом this будет доступна ссылка на сам создаваемый объект.

```
var obj = {
    a: 20,
    b: 30.3,
    sum: function(c){
        return this.a + this.b + c;
    }
}

console.log(obj.sum(7));
```



Работа с типами

Определяемые typeof типы:

- Number
- String
- Boolean
- Undefined
- Object
- Function

Типы, передаваемые по значению:

- Number
- String
- Boolean
- Undefined

Типы, передаваемые по ссылке:

- Object
- Function

1. Функции по сути являются объектами которые можно вызвать. Все остальное поведение аналогично объектам (за исключением **typeof** и функций конструкторов)

2. Стоит отметить, что если вы передаете объект со значением null в функцию, присвоение ему значения не приведёт к аналогичному результату в родительской области видимости.

Идентичность

- Во всех динамических языках помимо операций равенства, присутствуют операции идентичности.
- В случае сравнения на равенство сравниваются только значения, приводимые к общему типу.
- Идентичность подразумевает сравнение не только по значению, но и по типу. При этом преобразование типов не происходит.
- Операция идентичности и неидентичности записывается как ===,!==

Идентичность и равенство: Базовые типы

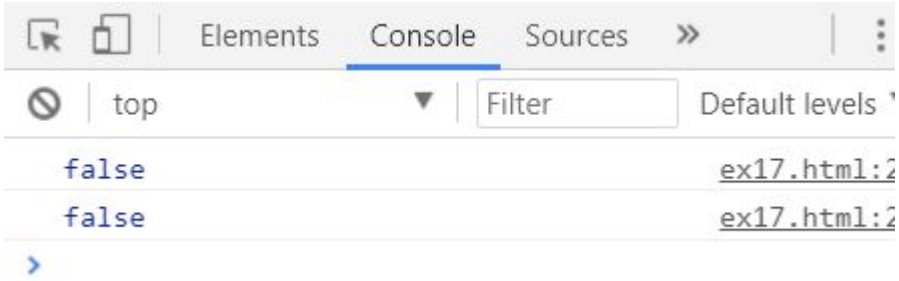
- При сравнении чисел на равенство все операнды приводится к числу.
- Пустые значения базовых типов равны между собой, и идентичны в случае совпадения типа.
- Специальные типы null и undefined равны между собой, но не равны с любыми другими значениями.

```
console.log("1" == 1); // true
console.log("0" == 0); // true
console.log("" == 0); // true
console.log(false == 0); // true
console.log(null == undefined); // true

console.log(false == undefined); // false
console.log(0 == undefined); // false
console.log(null == false); // false
console.log(null == 0); // false
console.log(null == ""); // false
console.log(null == "0"); // false
console.log(null == "null"); // false
```

Идентичность и равенство: Ссылочные типы

- Идентичность для ссылочных типов, таких как функции и объекты, не имеет смысла. При проверке на равенство проверяется полная идентичность объектов. Они должны ссылаться на оду и ту же область памяти.

```
var o1 = {
        a: 10
    },
    o2 = {
        a: 10
    };

function f1 (a){
    console.log(a);
};

function f2 (a){
    console.log(a);
};

console.log(o1 == o2);
console.log(f1 == f2);
```



Области видимости

- В отличии от стандартных C-подобных языков в JavaScript вложенные области видимости, создаваемые блоками, не ограничивают доступ к родительской области видимости.
- Пример с вложенными функциями.
- Для выделения памяти в определенной области видимости используется ключевое слово var.
- Объявление переменных без var приведёт к повреждению данных в родительских областях видимости или созданию переменной в глобальной области видимости.
- Основным объектом окружения является window. Именно в него записываются глобальные переменные.
- Переменные из глобальной области не подлежат сборки мусора.

Случайное определение переменной в глобальной области видимости

- Возьмите за правило в функциях объявлять все локальные переменные с помощью var, иначе вы можете записать ваши переменные в объект window и в результате логика приложения будет нарушена из-за некорректных данных.
- В примере, в первой функции, перед созданием переменной b, не было использовано ключевое слово var, соответственно, интерпретатор связал данную переменную с доступной аналогичной в глобальной области видимости. Если бы эта переменная отсутствовала, она была бы создана.

```
var b = 30;

function funcA(a)
{
    b = a + 10; // глобально b будет заменено
    console.log("funcA: " + b); // funcA: 30 + a
    return b;
};

function funcB(a)
{
    var b = a + 20;
    console.log("funcB: " + b);  // funcB: a + 20
    return b;
};

console.log(b); // 30
funcA(10);      // funcA: 20
console.log(b); // 20
funcB(20);      // funcB: 40
console.log(b); //20
```