Javascript part1

- DEV {Education}
- Преподаватель –Эльмар Гусейнов

Javascript part1

Plan of lecture

- JS History
- Variables
- Data types
- Operators
- Cycles
- Functions

Javascript (part1)

JS History

- JavaScript является высокоуровневым, динамическим, не типизированным и интерпретируемым языком программирования, который хорошо подходит для программирования в объектно-ориентированном и функциональном стилях. JavaScript был создан в компании Netscape на заре зарождения Веб технологий. Название «JavaScript» является торговой маркой, зарегистрированной компанией Sun Microsystems (ныне Oracle), и используется для обозначения реализации языка, созданной компанией Netscape (ныне Mozilla).
- С самого начала существовало несколько веб-браузеров (Netscape, Internet Explorer), которые предоставляли различные реализации языка. И чтобы свести различные реализации к общему стержню и стандартизировать язык под руководством организации ECMA был разработан стандарт ECMAScript. В принципе сами термины JavaScript и ECMAScript являются во многом взаимозаменяемыми и относятся к одному и тому же языку
- В 2009 году вышла новая важная версия стандарта языка под названием ECMAScript 5, а в 2015 - ECMAScript 6. Up to date - ECMAScript 10.

Javascript (part1)

Variables

- *Переменная* состоит из имени и выделенной области памяти, которая ему соответствует.
- Для объявления переменной в Javascript используются такие ключевые слова как var (ECMAScript 5) или let/const (ECMAScript 6) после которых будет идти идентификатор, который в дальнейшем будет являться именем нашей переменной:
- Пример инструкции объявления переменной (declaration statement):

  var/let(const) nameOfYourVariable

- С помощью такой инструкции мы можем объявить несколько переменных и проинициализировать их, инициализацией называется задание первичного значения для переменной:

Javascript (part1)

Variables

- **let variable = 20, anotherVariable = “some string”**
- Идентификатор объявления переменной может начинаться с буквы, нижнего подчеркивания или знака доллара, но первым знаком в идентификации переменных НЕ может быть цифра. Также в языке Javascript есть список зарезервированных ключевых слов, которые никак не могут являться данными идентификаторами (break, do, delete, case, continue, false etc...).
- Javascript, язык чувствительный к регистру, поэтому идентификатор myvariable и myVariable будут разными идентификаторами

Javascript (part1)

Data Types

На сегодняшний день в Javascript всего семь типов данных:

- Number (простой тип)
- String (простой тип)
- Boolean (простой тип)
- Symbol (простой тип)
- Null (специальный тип)
- Undefined (специальный тип)
- Object (объектный тип)

Javascript (part1)

Data Types

Первые шесть относятся к примитивным/простым типам данных и являются неизменяемыми (***immutable***), т.е. значения этих типов не могут быть модифицированными, изменить значение такого типа можно только путём перезаписывания нового значения в уже объявленную переменную.

Объявим переменные, присвоив значение каждого из этих типов:

var/let myNumber = 12345 (числовой литерал)

var/let myString = "some string" (строковый литерал)

var/let myBool = true (логический литерал)

var/let myNull = null (литерал null)

var/let mySymbol = Symbol()

var/let myUndef = undefined

В этой инструкции объявления все инициализирующие значения кроме Undefined и Symbol называются литералами соответствующего типа, тем временем как Undefined и Symbol это специальные идентификаторы.

Javascript (part1)

Data Types

Number. Числа в Javascript представляются 64-ёх битными значениями с плавающей точкой (т.е. хранятся в 64-битном формате IEEE-754, также известном как «double precision»). Все числа являются вещественными, отдельного типа данных для целых чисел нет.

String. Строки представляют собой цепочки 16-ти битных целочисленных значений. Каждый элемент занимает определённую позицию в строке, первый элемент имеет индекс 0, второй - 1 и т. д. Длина строки - это количество элементов в ней, отдельного типа данных для представления одного символа нет - для этого используется строка с длинной равной единице.

Boolean. Логический тип данных представляет результат логического выражения и может принимать одно из двух значений: истина (true) или ложь (false). Значение ЛЮБОГО типа в Javascript может быть преобразовано к логическому типу.

Javascript (part1)

Data Types

Symbol. Симбол - уникальный и неизменяемый тип данных, который может быть использован в качестве идентификатора для свойств объекта. Использование его в качестве ключа гарантирует уникальность данного ключа.

Два специальных типа Null и Undefined используются для представления единственного значения. Undefined означает что значение не присвоено, тогда как Null означает что значение присвоено, только оно пустое. К тому же Undefined является свойством глобального объекта, т.е. в глобальной области видимости существует переменная undefined равная undefined, тогда как Null - это литерал.

Javascript (part1)

Data Types

Object. Объект это коллекция свойств каждая из которых имеет свойство и значение (ассоциативный массив ключ-значение).

Для определения типа переменной используется специальный унарный оператор typeof, который возвращает значение типа указанной переменной в строковом представлении.

Необходимо помнить, что массивы, функции и объекты - это все составляющие ссылочных типов данных (Object), но:

typeof function a() {} ==> “function” (так удобно...)

typeof [] ==> “object”, но Array.isArray([]) ==> true (есть метод isArray)

typeof null ==> “object” (баг?)

Data types. Number

Числа, которые записываются прямо в коде программы называются числовыми литералами. Если говорить строго то отрицательных числовых литералов не существует, поскольку интерпретатор будет воспринимать знак минус как унарный оператор. Варианты записи чисел в разных системах исчисления:

Целочисленные формы записи:

console.log(10) - десятичная;

console.log(0xfffcc) - шестнадцатеричная; (можно ”x” писать в верхнем регистре);

console.log(0345) - восьмеричная;

console.log(012348) - десятичная, поскольку в восьмеричной нет цифр больше 7;

Data types. Number

Вещественные формы записи:

console.log(12.45)

console.log(.89) - если первым стоит 0 его можно пропустить

console.log(1.24e4) - экспоненциальная форма записи, может быть как с + так и с - (можно “e” писать в верхнем регистре);

Если математическая операция не может быть совершена, то возвращается специальное значение **NaN** (Not-A-Number):

console.log(0 / 0); console.log(Infinity / Infinity); console.log(Math.sqrt(-10))

Проверка NaN на равенство всегда возвращает false, даже когда мы пытаемся сравнить NaN с самим собой: console.log(NaN === NaN);

Data types. Number

Infinity – особенное численное значение, которое ведет себя в точности как математическая бесконечность ∞:

console.log(1 / 0);

console.log(-1 / 0); бывает и отрицательная бесконечность (-∞).

NaN и ***Infinity*** при попытке проверить их тип вернут “number” и если прогнать их через парсер - они будут являться идентификаторами, как тип данных ***Undefined***.

Data types. Number

Объект обёртка для number - **Number** (***функция-конструктор***):

*var/let **number** = 4000;*

console.log(typeof number); ==> "number"

*var/let **newNumber** = new Number(4000);*

console.log(typeof newNumber); ==> "object"

Далее мы можем воспользоваться методом для округления данного числового значения:

*console.log(**newNumber**.toFixed(2)); ==> "4000.00"*

*console.log(**number**.toFixed(2)); ==> "4000.00"*

Data types. Number

Понятие, что всё в Javascript - это объект базируется на данном примере, но не стоит забывать, что в данном случае, если это числовой литерал - интерпретатор создаёт для него объект обёртку на лету, вызывает метод ***toFixed***, а затем возвращает значение вызова функции при этом сохраняя первоначальный тип переменной, а сам объект ***Number*** уничтожается сразу после того как будет вычислено значение выражения.

Именно по данному примеру, можно сказать что простые значения **МОГУТ** вести себя как объекты но **НЕ ЯВЛЯЮТСЯ ОБЪЕКТАМИ!!!**

Arithmetic operators

Унарные:

унарный **“+”** и унарный **“-”** работают также как и на уроках математики; инкремент и декремент **“++”** и **“--”** - увеличивают или уменьшают значение переменной, бывают постфиксные и префиксные операторы инкремента и декремента:

```
let count = 10;
console.log(++count);
console.log(count);
console.log(count++);
console.log(count);
```

Arithmetic operators

Если поочерёдно выводить всё в консоль можно убедиться в следующем, что префиксный оператор сначала увеличивает или уменьшает значение на единицу, а затем возвращает значение, а постфиксный - сначала возвращает значение а затем уже увеличивает или уменьшает значение на единицу.

Бинарные:

“+”, “-”, “*”, “/”, “% (остаток от деления)” по поводу знака деления можно добавить, что в Javascript такой пример как **10/3** вернёт **вещественное число** а не целое.

Присваивание с операцией, рассмотрим следующие примеры где поймём некую возможную запись:

Arithmetic operators

let ***variable*** *= 100;*

variable *=* ***variable*** *+ 20;* ===> console.log(variable);

let ***anotherVariable*** *= 25;*

anotherVariable *=* ***anotherVariable*** ** 4;* ===> console.log(anotherVariable);

Всё это эквивалентно:

*console.log(****variable += 20****);*

*console.log(****anotherVariable *= 4****);*

Данное сокращение существует для всех арифметических бинарных операторов.

Arithmetic operators

Операторы отношения:

Возвращают значение логического типа true или false:

```
console.log(5 < 10);
console.log(5 > 10);
console.log(10 >=10);
console.log(8 <= 10);
```

Сравнение на равенство или неравенство:

```
console.log(10 === 10);
console.log(10 !== 10);
```

Arithmetic operators

Три знака “=” означает сравнение на равенство без приведения типов, когда знаков “=” только два - это означает, что сравнение на равенство будет происходит с приведением двух сравниваемых переменных к одному типу.

console.log(10 == “10”) ===> **true**

console.log.(10 === “10”) ===> **false**

Ошибки округления:

console.log(0.2 + 0.1) ===> ***0.30000000000000004....***

Про это нужно **НЕ ЗАБЫВАТЬ!** Это постоянная проблема работы с вещественными числами.

Data types. String

Строковые литералы записываются с помощью двойных или одинарных кавычек, а также в ES6 появилась форма записи строковых литералов с помощью косых кавычек:

console.log("string"); console.log('newString'); console.log(`anotherString`);

В языке Javascript всё это строки и разницы какие кавычки использовать нет, косые кавычки отличаются от других вариантов тем, что мы можем добавлять в данную строку с помощью специального синтаксиса **"${}"** некоторые данные, которые были определены выше по коду. Все варианты использования кавычек в кавычках и т.п. никуда не деваются, на помощь приходит понятие такое как **"Экранирование/Escaping"**.

dev {education}

Data types. String

console.log(“this is my string”); console.log(“this is my \”string\””);

Строка - это последовательность символов, каждый из которых занимает 2 байта, после выхода ES5 строки можно записывать на разных строках используя специальный символ \n, либо \t и т.п. Операторы “\n” и “\t” называются операторами последовательности в строковом литерале.

Оператор “+” - обозначает **конкатенацию**, в том случае если одним из его операндов является срока, **конкатенация** - это просто сложение строк.

Строки могут интерпретироваться как массивы:

let myString = “new string”;

console.log(myString[2]); - вместо charAt…

dev {education}

Data types. String. Methods

String.prototype.charAt()

возвращает указанный символ из строки

String.prototype.charCodeAt()

возвращает числовое значение Юникода для символа по указанному индексу (за исключением кодовых точек Юникода, больших 0x10000).

String.prototype.concat()

объединяет текст из двух или более строк и возвращает новую строку.

String.prototype.indexOf()

возвращает индекс первого вхождения указанного значения в строковый объект String, на котором он был вызван, начиная с индекса fromIndex. Возвращает -1, если значение не найдено.

Data types. String. Methods

String.prototype.toLowerCase()

возвращает значение строки, на которой он был вызван, преобразованное в нижний регистр.

String.prototype.toUpperCase()

возвращает значение строки, на которой он был вызван, преобразованное в верхний регистр.

String.prototype.trim()

удаляет пробельные символы с начала и конца строки. Пробельными символами в этом контексте считаются все собственно пробельные символы (пробел, табуляция, неразрывный пробел и прочие) и все символы конца строки (LF, CR и прочие).

Data types. Boolean

Логический тип может принимать только одно из двух значений: **истина** или **ложь**, в Javascript зарезервированы для этого два ключевых слова **true** и **false**. Обычно логические значения являются результатом операции отношения, например сравнения на равенство:

```
console.log(5 === 6); console.log(5 === 5);
```

Абсолютно любое значение в Javascript может быть преобразовано в логическое, для этого преобразования используется **конструктор Boolean** типа или унарный оператор **“!!”**.

Data types. Boolean

При таком подходе мы получим **false** только в случаях преобразования (Boolean(0), Boolean(-0), Boolean(“”), Boolean(null), Boolean(undefined), Boolean(NaN)), все остальные преобразования, даже [] или {} вернут true.

Логические операторы:

Логическое **&&** - возвращает истину только в том случае если оба его операнда истинны, то есть если первый операнд ложный - второй проверятся никогда не будет.

Логическое **||** - возвращает истину только в том случае, когда один из операндов истинный, то есть если первый операнд ложный - второй всегда будет проверятся.

Data types. Boolean

Таким образом можно писать так:

```
let number = 5;
console.log(number && 10 + number);
let prevString = "my String";
let newString = prevString || "default";
```

Унарный оператор логического отрицания: **"!"**, он возвращает инверсию текущего состояния переменной к которой применяется.

Data types. Null & Undefined

Два типа данных, которые обозначают отсутствие значения, **null** - значение пустое, **undefined** - значения вообще нет.

Варианты получения undefined:

```
let temp; console.log(temp);
let obj = {}; console.log(obj.property);
let ar = [1, 2, 3]; console.log(ar[3]);
const func = () => {return;}; console.log(func());
```

И если функция не возвращает никакого значения, то возвращаемым значением будет undefined.

Data types. Null & Undefined

При сравнении на равенство с приведением типа null и undefined - они равны, а без приведения - очевидно нет, так как типы у этих двух типов данных разные.

```
console.log(typeof null);
console.log(typeof undefined);
console.log(null == undefined);
console.log(null === undefined);
```

Data types. Symbol

Специальный тип данных, с помощью которого можно создавать уникальные идентификаторы свойств объектов. *Символы являются уникальными и неизменными значениями*. Позволяют создавать уникальные идентификаторы свойств объектов, при этом **НЕ резервируя** строковые названия для этих свойств.

Создание символа выглядит следующим образом:

Функция символ **НЕ является конструктором**, поэтому мы не можем использовать ключевое слово new.

```
let symbol = Symbol(); console.log(symbol); ===> Symbol();
console.log(typeof symbol); ===> "symbol";
```

dev {education}

Data types. Symbol

У функции символ есть один необязательный параметр, который можно передать для последующего наименования созданного символа, а также это может быть удобно для отладки.

```
let symbol = Symbol("name");
console.log(symbol); ===> Symbol(name);
let anotherSymbol = Symbol("name");
console.log(anotherSymbol); ===> Symbol(name);
console.log(symbol === anotherSymbol); ===> false;
```

Проверка на равенство двух одинаково созданных символа вернёт ложь, потому что каждый символ уникален (like hash code).

Data types. Symbol

Способ создания символа с помощью метода “for”:

```
let symbol = Symbol.for(“name”);
console.log(symbol); ===> Symbol(name);
let anotherSymbol = Symbol.for(“name”);
console.log(symbol === anotherSymbol); ===> true;
```

Также благодаря методу **“for”** символ можно получить с любого места в программе. Так как они заносятся в так называемый **глобальный реестр**. Соответственно символы, созданные **БЕЗ** использования метода **“for”** - **НЕ** заносятся в глобальный реестр.

Data types. Symbol

Рассмотрим метод **“keyFor”**:

```
let symbol = Symbol.for(“name”);
let name = Symbol.keyFor(symbol);
console.log(name); ===> name;
```

Если же методу “for” не передать имя, то мы получим undefined в качестве имени символа.

Использование символов:

В ES5 свойства объектов должны быть строками, в ES6 - помимо строк они могут быть символами. В Javascript есть встроенные символы **“WellKnown Symbols”**, например **Symbol.iterator**, ...etc;

Data types. Symbol

```
let user = { userName: "Vasya", [Symbol('password')]: "asdf" }
console.log(user.password) ===> undefined;
console.log(Object.keys(user)) ===> [userName];
console.log(Object.getOwnPropertyName(user)) ===> [userName];
let password = user[Symbol.for('password')];
console.log(password); ===> undefined;
```

Потому что данный символ мы создали **БЕЗ** метода **"for"**. Если же переписать - тогда мы получим значение данного свойства.

Data types. Symbol

В ES6 был добавлен ***специальный метод*** для того чтобы посмотреть символы объекта: *console.log(Object.**getOwnPropertySymbols**(user));*

Если же мы уберём название символа то в консоле мы увидим только лишь то, что у объекта есть символ, но как достучаться до него мы не будем знать. Есть вариант сохранения такого анонимного символа в переменную, для того чтобы иметь к нему доступ:

```
let password = Symbol();
let user = { userName: "Vasya", [password]: "asdf" }
```

Символы были добавлены в ES6 не для того чтобы что-то спрятать, а именно для того, чтобы *избежать конфликта имён свойств*.

Data types. Objects

Объекты в Javascript это набор свойств которые представляют собой пару: имя - значение, они разделяются двоеточиями и перечисляются через запятую (в др. языках аналогами являются - ассоциативный массив, или словарь).

{ name: “Dima”, age: 28 } - объектный литерал, литерал это один из способов создания объекта.

Мы можем записать этот объект в отдельную переменную (*let obj = ...*) и обращаться к свойствам (или полям) этого объекта с помощью выражения обращения или обращения доступа, которые имеют два варианта обращения - **obj.name ^ obj[name]**.

Data types. Objects

С помощью такого обращения к свойствам объекта мы можем изменять текущие свойства или добавлять новые на лету.

Значением любого свойства может быть функция и в таком случае такое свойство называют методом.

Способы создания объекта:

Самый древний, с использованием **функции конструктора и оператора new**:

```
let newObject = new Object(); newObject.name = 'Fred';
```

Но сейчас объектный литерал выглядит лучше (**{ key: value }**).

Data types. Objects

Следующий вариант создания объекта в Javascript является статический метод **create** класса **Object**, который на вход принимает *Прототип* на основании которого будет создан новый объект.

*let **newObject** = Object.**create**({x: 20, y: 30});* *вместо* ***{x: 20, y: 30}*** мог быть и **null** если мы создаём объект без наследования свойств какого-либо прототипа.

Если после такого создания мы захотим проверить если у нашего объекта такое свойство как "x" с помощью метода **hasOwnProperty**, то результатом будет **false**, поскольку это свойство принадлежит прототипу.

Data types. Objects

Если мы создадим свойство у нового объекта с таким же именем, то тогда мы получим true:

```
var newObject = Object.create({x: 20, y: 30});
newObject.x = 40.
console.log(newObject.hasOwnProperty('x')) ===> true;
```

Если мы обратимся к такому свойству - вернётся родное а не наследованное:

```
console.log(newObject.x) ===> 40;
```

Для удаления свойств объекта существует унарный оператор **delete**:

```
delete newObject.x; console.log(newObject)
```

Data types. Objects

Выведет объект прототипа, так как одно единственное свойство мы удалили.

delete - удалит только то свойство, которое принадлежит объекту, а не прототипу. Чтобы удалить в данном примере свойство “x” необходимо обратиться напрямую к самому прототипу.

Для проверки наличия свойства в объекте существует бинарный оператор **“in”.** Ему всё-равно, наследованное это свойство или родное, потому что если нет - пропадает смысл работы метода **hasOwnProperty**:

console.log('x' in newObject); ===> true

console.log.('z' in newObject); ===> false

Data types. Objects

Очень часто используют выражение обращения (если свойство есть то вернётся **значение**, если свойства нет - то вернётся **undefined**):

console.log(newObject.x) ===> 20; console.log(newObject.z) ===> undefined

Оператор **“in”** различает отсутствующие свойства и свойства которые были установлены в **undefined** *вручную*. Проверим данное заключение на примере:

*console.log(**newObject**.z) ===> undefined;*

*console.log('z' in **newObject**) ===> false; **newObject**.z = undefined;*

*console.log(**newObject**.z) ===> undefined;*

*console.log('z' in **newObject**) ===> true;*

Data types. Operators

Побитовые операторы.

Побитовое И (AND)

a & b

Ставит 1 на бит результата, для которого соответствующие биты операндов равны 1.

Побитовое ИЛИ (OR)

a | b

Ставит 1 на бит результата, для которого хотя бы один из соответствующих битов операндов равен 1.

Data types. Operators

Побитовое исключающее ИЛИ (XOR)

`a ^ b`

Ставит 1 на бит результата, для которого только один из соответствующих битов операндов равен 1 (но не оба).

Левый сдвиг

`a << b`

Сдвигает двоичное представление a на b битов влево, добавляя справа нули.

Побитовое НЕ (NOT)

`~a`

Заменяет каждый бит операнда на противоположный.

Data types. Operators

Правый сдвиг, переносящий знак

a >> b

Сдвигает двоичное представление a на b битов вправо, отбрасывая сдвигаемые биты.

Правый сдвиг с заполнением нулями

a >>> b

Сдвигает двоичное представление a на b битов вправо, отбрасывая сдвигаемые биты и добавляя нули слева.

Тернарный оператор. *(условие ? выражение если **true** : выражение если **false**)*

Data types. Operators

Есть большая разница между выражениями:

*if (false **&&** console.log(1)) и if (false **&** console.log(1));*

*if (true **||** console.log(2)) и if (true **|** console.log(2));*

Пример 48^23**:**

32	16	8	4	2	1
1	1	0	0	0	0
0	1	0	1	1	1

Затем побитово необходимо сложить каждый из разрядов и затем необходимо найти остаток от деления на 2 каждого из разрядов.

*1 2 0 1 1 1 ===> 1 0 0 1 1 1 ===> Ответ **39**.*

dev {education}

Data types. Cycles

Условные инструкции (**switch(case, break, default, return), if/else(else if**)).

Циклы (P.S.движение к нулю будет быстрее!).

FOR:

for(“инициализация”; “условие”; “инкремент/декремент”) тело цикла.

for(;;); - бесконечный цикл который делает ничего;

for(;;) console.log(‘a’) - бесконечный цикл который выводит a;

for(var i = 0; i < 10; i++) console.log(i) ===> 0 - 9.

for(var i = 10; i--) console.log(i) ===> 9 - 0.

Data types. Cycles

WHILE:

let i = 0;

while(i < 10) console.log(i++); ===> 0 - 9.

while(i--) console.log(i++) ===> 9 - 0.

DO_WHILE:

Условие цикла будет выполняться хотя бы один раз вне зависимости от истинности выражения.

FOR_IN:

Может применятся как и для объектов так и для массивов:

*for(let key in **array**^object) console.log(**array**[key]^**object**[key]);*

Data types. Cycles

Где для массива **keys** это будут индексы, а для объекта - ключи (поля, свойства) и при обращении array[key] ^ object[key] мы будем получать значение этих свойств или данных массива.

FOR_OF:

добавлен в ES6:

for(var item of array) - с объектом не пройдёт, так как объект не имеет специального свойства **[Symbol.iterator]()**;

Data types. Functions

Функция - это количество инструкций, которые определяются единожды и потом могут быть вызваны несколько раз. Если в Вашем скрипте Вы повторяете одинаковые строчки кода несколько раз, то это уже говорит о том, что было бы неплохо вынести эти строчки в отдельную функцию. Функция, которая создаёт новый объект называется **конструктором**.

Варианты создания функции:

Function declaration:

```
function myFunc(name) {
    return "Hello" + name;
}
```

dev {education}

Data types. Functions

Function expression:

```
var myFunc = function(name) {
    return “Hello” + name;
}
```

Основное отличие между ними: функции, объявленные как **Function Declaration**, создаются интерпретатором до выполнения кода, поэтому их можно вызвать до объявления, например.

Анонимные функции:

```
() => console.log(‘launch anonymous function!!!’)
```

Data types. Functions

Функциональное выражение, которое не записывается в переменную, называют анонимной функцией.

new Function(params, code)

params - параметры функции через запятую в виде строки.

code - код функции в виде строки.

```
let sum = new Function('a,b', 'return a+b;');
let result = sum(1, 2);
console.log(result); ===> 3
```

Arrow Functions. (синтаксический сахар) подробнее с ней познакомимся на теме с ES6, основная идея - динамическая привязка контекста. **dev** {education}

Data types. Functions

Доступ к аргументам функции можно получить с помощью ключевого слова **arguments** внутри функции. Это массив, содержащий все аргументы которые были переданы в функцию, вне зависимости сколько реально указано возможных аргументов; **arguments** не является массивом. Он подобен массиву, но не имеет никаких свойств массива, кроме length. Но можно превратить в настоящий массив (ES6 args = [...arguments]).

Callback функция - это функция обратного вызова.

typeof yourFunc - всегда вернёт **“function”**.

typeof String - вернёт **“function”**.

typeof new String() - вернёт **“object”**.