**我是如何理解”Another JavaScript quiz”中的题目**

### 一、你这个变态，给我滚开

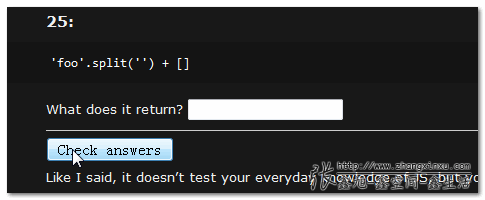
时光冉冉，或多或少见过一些JavaScript相关的题目，其中很多属于变态级别的！各种奇怪符号写法拼在一起、尼玛还有兼容性问题，估计道格拉斯都不知道答案。

对于这种整得亲妈都不认识的变态问题，实际上是没有什么参考价值的。好比要考察外星人对人类的了解，结果你那下面这货来做测试，看到亲戚的外星人一定会云里雾里的，但有意义吗？  


好的JavaScript测试题目应该是：门外汉见了皱眉，行家见了疑惑题目是不是简单了点，同时考察点覆盖全面。//zxx: 我目前还没有这个能耐设计出如此优秀的题目。

这里要介绍的”[Another JavaScript quiz](http://james.padolsey.com/javascript/another-javascript-quiz/)“(by james)中的题目不是属于变态题目，而是确实属于变态题目，不过是表面上的，很多内容确实可能会遇到的。综合评价下就是：面试价值不及格，学习价值是很赞的，因此，探讨分享很有意义。

“[Another JavaScript quiz](http://james.padolsey.com/javascript/another-javascript-quiz/)“中有25个很简洁的JavaScript测试题，全部都是考察返回值的，例如1 && 3的返回值是？

25道题目全部完成后，可以点击下图所示的按钮，检查你的正确率以及那些题目出错了。  


我自己快速测试了下，如果每题算4分的话，我的[成绩是56分](http://weibo.com/1263362863/ztn6GCI0Z)，不及格，可见自己对JS的学习以及理解还有一段路要走。

您也可以做一下这些题目，完成之后，若有疑惑，可以参考我下面纯个人的理解 – 部分理解可能不准确，欢迎指正。同时，网上应该有其他一些前辈的解释，您也可以参考参考。

### 二、公子，不急，慢慢来嘛

1. 1 && 3 //返回的是？

结果是3.

&&是几乎无人不知，表示“与”。男方要娶水灵灵的妹子，七大姑八大婆都要同意，否则继续锻炼右手吧。

上面1 && 3本质上等同于大姑妈 && 三姑妈。

从左往右，如果“大姑妈”通过，继续“三姑妈”；否则直接返回“大姑妈”。因为“三姑妈”后面没有其他亲戚了，因此直接返回“三姑妈”。

在JavaScript中， 1 == true // true，因此，1 && 3等同于“大姑妈”通过了，返回最后一个检测的亲戚“三姑妈”，也就是这里的结果3.

因此，实际上，我们平时的 if (1 && 3) {} 等同于 if (3) {}.

如果这里的题目换成0 && 3返回的是？那结果是如何呢？

因为0 == false // true, 因此走不到“三姑妈”这一关，直接返回了0. 也就是if (0 && 3) {} 等同于 if (0) { /\* 不会执行 \*/ }.

**实用性**  
可以避免if嵌套。例如，要问页面上某个dom绑定点击事件，我们需要先判断这个dom元素存不存在，我们可能会这样做：

var dom = document.querySelector("#dom");

if (dom) { dom.addEventListener("click", function() {}); }

实际上，我们可以使用&&做一些简化：

var dom = document.querySelector("#dom");

dom && dom.addEventListener("click", function() {});

1. 1 && "foo" || 0 //返回的是？

很简单的一道题，结果是"foo".

这里出现了一个新的关系符||, 表示“或”的意思。男方要娶干巴巴的妹子，七大姑八大婆只要一个说好就可以了。比方说1 || 3表示：如果大姑妈说可以，则成了，返回大姑妈；如果大姑妈不允许，再看看三姑妈的意思。

因此，if (1 || 3) {}实际等同于if (1) {}; if (0 || 3) {} 等同于if (3) {}.

**实用性**  
||可以让我们使用一种更快捷简易的方式为参数添加默认值。例如，写jQuery插件的时候，可选参数是可以缺省的，此时实际上值为undefined，会让后面的参数extend产生困扰。因此，我们会经常见到类似这样的代码：

$.fn.plugin = function(options) {

options = options || {};

// ...

};

显然，根据上面的理解，1 && "foo"返回的是"foo", 而"foo" || 0, 因"foo" == true为true, 自然返回是“大姑妈” – "foo". 这就是最终结果"foo"的由来。

1. 1 || "foo" && 0 //返回的是？

结果是1.

据说逻辑与运算符(&&)优先级要大于逻辑或(||)。如果是这样，那这里的结果返回应该是这样子的：1 || "foo" && 0 → 1 || 0 → 1.

1. (1, 2, 3) //返回的是？

结果是3.

这里考察的是逗号运算符，也称多重求值。逗号运算符据说是优先级最低的运算符。  
一般表现形式是：大姑妈，二姑妈，三姑妈，……N姑妈。  
运算规则如下：折腾大姑妈，折腾二姑妈，折腾三姑妈，……折腾N姑妈，最后返回的N姑妈的返回值。

因此，这里1, 2, 3就是折腾了1，折腾了2，折腾了3并且返回了3的返回值。因此，结果是3.

好，现在提问：alert(1, 2, 3);的弹出值是？

结果是3……………………是不可能的！正确结果是**1**.

不要困扰。原因很简单，alert身后的()实际上也是一种运算符，这里指函数调用，优先级相当的高。这里，1, 2, 3已经被奴役成alert方法（弹出第一个参数值）的参数了，因此，弹出的是1.

如果是alert((1, 2, 3));则弹出的就是3了。

**实用性**  
举个很简单的例子，i, j两个变量同时递增，可能我们会这样写：

var i=1;

var j=1;

i+=1;

j+=2;

我们可以借助逗号运算符获得更好地阅读体验，至于性能是否有提高，我个人并不清楚，就算有，差异也是可以忽略的。

var i=1, j=1;

i+=1, j+=2;

1. x = { shift: [].shift };
2. x.shift();
3. x.length; //返回的是？

结果是IE6/IE7 undefined, 其他浏览器0. 不过，个人观点，IE6/IE7其实并不属于这个题目的考察浏览器。因此，我们着重讨论为何IE8+等浏览器为何结果是0.

首先，我们了解下数组的shift方法。 我个人一般把pop/push归为一对基友，移除尾部元素和尾部添加元素，都是做扫尾工作的。shift/unshift当作另外一对基友，移除头部元素和头部添加元素，都是在头部打点的。记住，单词字符个数少的(*pop*,*shift*)都是做删除的，这样记忆就不会混淆了。

从外表上看，shift方法移除数组中的第一个元素并返回这个元素。如果这个数组是个空数组，则返回undefined. 一般而言，shift操作会改变后面所有数组项在内存中的地址，因此，相比pop方法要慢得多。

从内在来看，shift还能给对象自动添加length属性。Mozilla开发者网络(MDN – Mozilla Developer Network)的解释[是这样子的](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Array/shift)：

shift is intentionally generic; this method can be called or applied to objects resembling arrays. Objects which do not contain a length property reflecting the last in a series of consecutive, zero-based numerical properties may not behave in any meaningful manner.

简体中文表示就是（这里的释义自己只有80%确认，若有不准确，请极力指证http://mat1.gtimg.com/www/mb/images/face/30.gif）：  


shift可以有意泛化（变身成鸭子）；该方法可以被类数组对象call或者apply. 对象如果没有length属性，可能会以无意义的方式在最后反射一系列连续，基于0的数值属性。

什么意思呢？我们看下面这个更容易理解的问题：

x = {};

[].shift.call(x);

x.length; //返回的是？

结果是0. //zxx: IE6/IE7 undefined.

x对象原本木有length属性，在被数组shift方法call后，添加了一个值为0的length属性。此为数组shift方法的泛化性，专业术语为泛型(generic)。其基友方法，例如pop, push等都是如此。

OK, 回到原题，实际上，x.shift()的调用等同于[].shift.call(x), 不明白？看下面的一步一步分析。

var x = {

shift: function() {

console.log(this === x); // true

}

};

x.shift();

因此，在

x = { shift: [].shift }

条件下，  
x.shift执行的就是[].shift的执行，只不过，[].shift()函数中的this上下文就是x(因为this===x)，就等同于直接的[].shift.call(x)调用。

这条题目中x对象的shift属性名实际上是用来干扰，提高解答难度的刻意命名。我们使用其他命名，结果也是一样的。

x = { shit: [].shift };

x.shit();

x.length; // 0

1. {foo:1}[0] //返回的是？

结果是[0], 或者这种表现形式0 { 0 : 0 } – 来自IE控制台.

不要试图使用alert或者控制台console.log输出，这只会返回不一样的结果undefined，哦？为何会有这等差异？

出题者james在”[Labelled blocks, useful?](http://james.padolsey.com/javascript/labelled-blocks-useful/)“中有这样的解释：

Since JavaScript doesn’t have block scope, using a block anywhere other than in the conventional places (if/while etc.) is almost totally pointless. However, as I mentioned, we could use them to annotate and contain logically related pieces of code…

意思是说：

因为JavaScript没有块作用域，所以，如果语句块不是常规使用，如if/while等，其几乎就是打酱油的。甚至，我们可以利用这个特性注释或者包含相关的逻辑片段代码…

我们有必要好好理解这里“打酱油的”意思，这里的“打酱油”并不是指{}块中语句是打酱油，而是其本身就是个酱油。嘛意思，实例说明一切：

2 // 返回值为2

{2} // 返回值为2

str = "string" // 返回值为string

{ str = "string" } // 返回值为string

foo: 1 // 返回值为1

{ foo: 1 } // 返回值为1

也就是说花括号几乎就是皇帝的新衣。因此，这里的答案就不难理解了，{foo:1}[0]实际上就是foo:1; [0]. 返回的就是[0]本身。

注意这里反复出现的措辞“几乎”。“几乎酱油”的潜台词是有时候还能顶个臭皮匠。james举了个在块中使用break语句的例子，如下：

var x = 1;

foo: {

x = 2;

break foo;

x = 3;

}

x === 2; // true

非这种情况的break/continue只能在switch语句以及循环中使用。很有意思吧，我反正是学习了http://mat1.gtimg.com/www/mb/images/face/30.gif！

1. [true, false][+true, +false] //返回的是？

结果是true.

首先，我们了解下+true和+false是个什么东西。

想必都清楚+1与-1是什么东东。指的是正数1与负数1. 同时，稍微对JS有了解的人也清楚，true == 1,false == 0. 因此，实际上，[+true, +false]就是[+1, +0].

[true, false]为数组，后面的[+true, +false]实际为索引，然而索引只需要一个值，因此，[+true, +false]返回的实际是我们上面提到的逗号运算——返回最后一个值，也就是+0, 也就是0.

因此，本题的问题其实是：

[true, false][0] //返回的是？

旺财估计都已经明白了，就不多说了。

1. ++'52'.split('')[0] //返回的是？

结果是6.

此题难点在于运算符的优先级，看来有必要把运算符优先级都展示下。参见下表（数据取自[美拓blog](http://blog.meituo.net/2010/08/29/%E4%B8%8D%E5%8F%AF%E4%B8%8D%E7%9F%A5%E7%9A%84javascript%E8%BF%90%E7%AE%97%E7%AC%A6%E4%BC%98%E5%85%88%E7%BA%A7/)）：

|  |  |  |
| --- | --- | --- |
|  | **运算符** | **描述** |
| **1** | . [] () | 字段访问、数组下标、函数调用以及表达式分组 |
| **2** | ++ — – ~ ! delete new typeof void | 一元运算符、返回数据类型、对象创建、未定义值 |
| **3** | \* / % | 乘法、除法、取模 |
| **4** | + – + | 加法、减法、字符串连接 |
| **5** | < < >> >>> | 移位 |
| **6** | < <= > >= instanceof | 小于、小于等于、大于、大于等于、instanceof |
| **7** | == != === !== | 等于、不等于、严格相等、非严格相等 |
| **8** | & | 按位与 |
| **9** | ^ | 按位异或 |
| **10** | | | 按位或 |
| **11** | && | 逻辑与 |
| **12** | || | 逻辑或 |
| **13** | ?: | 条件 |
| **14** | = += -= | 赋值、运算赋值 |
| **15** | , | 多重求值 |

从上表可以看出点.以及[]要比++的优先级高。因此，这里的问题等同于：

var result;

result = "52".split(""); // ["5", "2"]

result = result[0]; // "5"

result = ++result; // 6

**注意：**  
虽然说++["5", "2"][0]的返回值是6, 但是直接++"5"或者++5却是会报错的（FireBug控制台显示“invalid increment operand”错误），据说是因为只有变量才能++ 而'5'只是一个字符串。

1. a: b: c: d: e: f: g: 1, 2, 3, 4, 5; //返回的是？

结果是：5.

这里需要讲下标记语句(Labelled Statements). 下文为我理解的[ECMAScript文档中的解释](http://ecmascript.cn/#232)：

语句可以有个标签前缀。标记语句只和break或continue标记的语句结合使用（想想第7题最后那个break的例子）。ECMAScript中并无goto语句。

ECMAScript程序中，如果标记语句的标签标识符有重复，则会出错。比方说a: {a: 1};就会嗝屁(FireBug显示SyntaxError: duplicate label错误)，而a: {b: 1};继续逍遥快活。但是，这并不适用于标记语句中或嵌套或直接或间接FunctionDeclaration体中出现的标签。如a: (function() {a: 1});就是OK的。

生产标识符(Identifier)：语句的执行是通过给标签集增加标识符，然后再执行语句。如果标签语句本身就有一个非空的标签集（冒号后面的语句还有类似结构，例如a: b: 1，b:1为a标签集的标签语句，但这个语句本身就含有一个标签集），则在语句执行之前这些标签们添加到标签集之中。如果语句执行的结果是(break, V, L)，其中L等于标识符，则产生的结果是(normal, V, empty). //zxx: 标识符实际就是指变量名、函数名、数组名等. 此处我是看不懂的，因此无法具体解释

执行标记语句(LabelledStatement)之前，其包含的语句被认为具有一个空的标签集，除非他是一个迭代语句或switch语句。在这种情况下,它被视为拥有一个单一元素组成的空标签集。

上面这点东西折腾了个把小时都没搞清楚要说的是什么，貌似也不能很好解释这里的结果，投入与产出比太低，郁闷！只怪功力不够，或许过两年再过来看，就很轻松了。

后又搜索了下，发现[MDN的解释](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/label)要通俗的多：  
所谓标记语句，语法如下：

label :

statement

就是：

标签 :

语句

其中label可以是JavaScript中任意非保留关键字的标识符；statement中break可以用在任何标记语句中，continue用在循环标记语句中。

**MDN建议不使用labels**  
labels在JavaScript中并不常用，因为其降低了可读性以及易理解性，尽可能避免使用labels标记，根据不同情况，选择函数调用或者抛出错误。

下面为一个典型的标记语句的例子：

var i, j;

loop1:

for (i = 0; i < 3; i++) { // 首页语句标记是 "loop1"

loop2:

for (j = 0; j < 3; j++) { //第二个语句标记为 "loop2"

if (i == 1 && j == 1) {

continue loop1;

} else {

console.log("i = " + i + ", j = " + j);

}

}

}

// 输出是:

// "i = 0, j = 0"

// "i = 0, j = 1"

// "i = 0, j = 2"

// "i = 1, j = 0"

// "i = 2, j = 0"

// "i = 2, j = 1"

// "i = 2, j = 2"

// 注意是如何不输出 "i = 1, j = 1"和"i = 1, j = 2"的

扯了这么多，回到问题本身。

根据规范，a: b: c: ... g:在语句执行之前会归到一个标签集中，为一个集合。因此，走个极端的话，我们可以这么理解：  
abcdefg: 1, 2, 3, 4, 5. 1, 2, 3, 4, 5一开始有说明的逗号多重运算啦——返回最后一个值，因此，本题就类似于提问：

abcdefg: 5 // 返回的是？

1. {a: 1, b: 2}[["b"]] //返回的是？

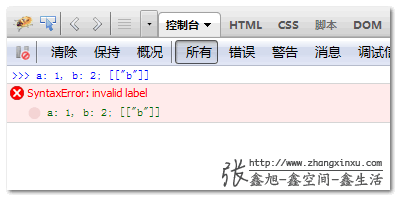
FireBug控制台显示：SyntaxError: invalid label.

得知是这个结果的时候，我的懵懂硕大的眼睛立马无语成了一条缝，￣﹏￣

第6题中，我们已经讲过，JavaScript没有块作用域，块本身几乎是个酱油，因此这里的测试题实际等同于：

a: 1, b: 2; [["b"]]

结果显示：SyntaxError: invalid label.



现在的疑问是，为何a: 1, b: 2会报错？微博提问……

30分钟后，[@紫云妃](http://weibo.com/1708684567)给出了这样的解释：

逗号运算符右侧必须是个表达式,不能是非表达式的语句,这个例子中a: 1, b: 2的右侧的b: 2是一个LabelledStatement, 不是表达式。

我觉得是相当靠谱的回答。

1. "b" + 45 //返回的是？

尼玛是不是感动地眼泪哗啦的。这种感觉就像是被后妈饿了个把月，在今天这个阳光明媚的日子赏了一块红烧肉，一下子感动地眼泪喷泉般涌出来。

结果是："b45".

字符串+数值=字符串。

1. {a:{b:2}} //返回的是？

结果是：2.

自从翻越了珠穆朗玛，才知道原来佘山就是个小土包。这个问题已经是小菜了，JavaScript没有块作用域，因此，{a:{b:2}}近乎于a:b:2, 想起a,b,c,...g的例子没有，显然，这里返回值是2.

1. (function(){}()) //返回的是？

结果是：undefined.

函数如果是空，或者没有指定返回值，则返回的是undefined.

关于此题，我在眼睛微博上抛出了这么个问题：(function(){}())和(function(){})()这两种写法有什么区别呢？

1个小时过去了，以下几个评论比较见血：

[@川川哥哥勤劳致富](http://weibo.com/cohlint1q81) 效果是一样的,周爱民那本书上提到了语法解读上的区别,不过感觉讲得不是很清晰

[@貘吃馍香](http://weibo.com/itapir) 括号 匿名函数表达式执行 括号 跟 括号 函数表达式定义 括号 => 返回匿名函数句柄执行 的区别……

[@\_Franky](http://weibo.com/1900199050) 抽象语法树 和 运行时 无差异. 除非这个解释引擎脑残了. 因为分组运算符 “()” 在生成语法树的过程中被消除了. <http://t.cn/zWTxgKa> 这篇我有顺道提到过这个问题

1. [1,2,3,4,5][0..toString.length] //返回的是？

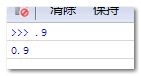
结果还是：2.

本题涉及的知识点挺多的。

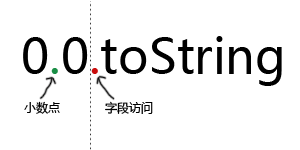
**①**首先是诡异的连续两个点... 我们在写CSS的时候，常常会有这样的写法：

font-size: .9em;

color: rgba(0, 0, 0, .35);

点前面如果是0则可以自动缺省。这条规则似乎在JavaScript中也是适用的，比方说.9，返回的就是0.9.  


因此，这里0..toString实际上等同于0.0.toString. 下面有个疑问了，为什么不直接0.toString而是0.0.toString要多搞出一个0呢？据说是这样子的，数值后面的点·有两种作用，一种就是当作小数点适用，另外就是用做字段访问。显然，0.toString这里的点会被当作小数点，于是，直接报错了！写成(0).toString可以避免此问题。再来看看0.0.toString, 显然，再做解析的时候，0.0中的点当作了小数点，显然，0.0后面的那个点没有任何理由再被当作小数点了（小数不可能有两个小数点的），而是表示字段访问。于是，万事大吉，阳光明媚（下图版权所有）！



**②**下面看看toString, toString为JS内置方法，ECMAScript规范上转换规则如下：

|  |  |
| --- | --- |
| **输入类型** | **结果** |
| Undefined | “undefined” |
| Null | “null” |
| Boolean | 如果参数是 true，那么结果为 “true”。 如果参数是 false，那么结果为 “false”。 |
| Number | 结果等于**输入的参数**（不转换）。 |
| String | 参见下文的文法和注释。 |
| Object | 应用下列步骤：   * 1. 调用 ToPrimitive(输入参数, 暗示 字符串类型)。   2. 调用 ToString(Result(1))。   3. 返回 Result(2)。 |

从上表可以看出，数值使用toString方法是有参数的。

在JavaScript中，这个参数仅1个，称之为“基”，决定了数值转换的进制大小。默认情况下是10进制。例如：

(10).toString(); // "10"

我们还可以设置“基”为2进制，8进制或16进制，则结果大变样：

(10).toString(2); // "1010"

(10).toString(8); // "12"

(10).toString(16); // "a"

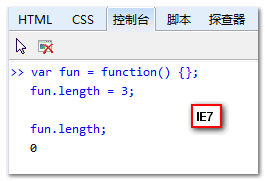
在[实现颜色值16进制与rgb转化](http://www.zhangxinxu.com/wordpress/?p=646)的时候，这些基数参数就很管用。

**③**最后了解下函数的length与参数个数，看这个：(function(){}).length的结果是0. 为何？  
每个function函数似乎都有一个**不可写**的length属性，对应这个函数的参数个数。

var fun = function() {};

fun.length = 3;

fun.length; // 0



**④**现在，回到我们的问题，事情就豁然开朗了：[0..toString.length]实际上就是数值(0.0)应用toString方法的参数个数是多少？根据上面描述，数值使用toString转换参数个数为1。于是，本题结果就是[1,2,3,4,5][1]=> 2.

1. ({} + 'b' > {} + 'a') //返回的是？

结果是：true.

需要注意最外面的括号。如果没有最外面的括号，则{}则几乎无意义，但是这里，作为常规用法，{}表示原生对象。因此，这里的比较实际上就是比较("[object Object]b" > "[object Object]a"), 因此返回的是true.

说点题外的，如果最外部没有括号，{} + 'b'返回的是NaN. 于是{} + 'b' > {} + 'a'变成了比较NaN > NaN, 结果为false.

1. Number.prototype.x = function(){ return this === 123; };
2. (123).x(); //返回的是？

结果是：false.

我们这里使用了严格相等。实际上this和123属于不同的类型。

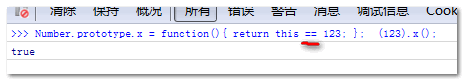
typeof this === “object”

typeof 123 === “number”

因此，结果为false. 如果我们把题目修改成弱等于，则返回结果就是true了，见下截图：

Number.prototype.x = function(){ return this == 123; };

(123).x(); //返回的是？



1. Array(2).join() //返回的是？

结果是：","

Array(2)的返回值是[undefined, undefined]，因此，其使用join连接之后，就是个逗号","（数组join为指定连接符时候使用默认的逗号","）。

这里我要抛出一个微博上没有满意解答的问题：Array(2)和new Array(2)的区别在什么地方呢？ 求指点迷津！

1. vars: var vars = vars; //返回的是？

结果是：undefined.

现在看此题就简单多了，标记语句，返回值就是var vars = vars的返回值undefined.

var vars = vars并不会报vars为定义的错误是在于JS的置顶解析，其实var vars = vars的运作是这样子的：

var vars;

vars = vars;

1. { foo = 123 } //返回的是？

结果是：123.

花括号不在括号中、if语句或者循环中，属于非常规酱油用法，形同虚设，问题等同于foo = 123的返回值是？

1. x = 1; (function(){return x; var x = 2;}()) //返回的是？

结果是：undefined.

此处考察的是JavaScript的“预解析(hoisting)”，也称“置顶解析”，我之前有[翻译过相关文章](http://www.zhangxinxu.com/wordpress/?p=1162)。

所谓“预解析”指的是在当前的作用域内，无论在哪里变量声明，在幕后，其都在顶部被“预解析”了。因此，本题的实际“解析”是：

x = 1; (function(){var x; // 此时x为undefined

return x; x= 2;}())

因此结果是undefined.

1. delete [].length; //返回的是？

结果为false.

delete用来删除对象属性，成功删除返回true, 如果对方防御很强删不动则返回false.

数组中的length属性是不可删除的，因此这里返回的是false.

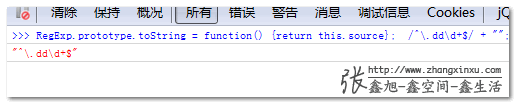
delete相关的知识点是很多的，比说法window.x = 1这里的x可以被delete. var x =1这里的x就不能被delete. 图灵社区有篇不错的译文：“[理解delete](http://www.ituring.com.cn/article/details/7620)”，推荐阅读。

1. RegExp.prototype.toString = function() {return this.source};
2. /3/-/2/; //返回的是？

结果是：1.

正则表达式有如下一些属性：source, global, ignoreCase, multiline, lastIndex. 其中：source属性为构成正则表达式Pattern的字符串；global属性是一Boolean值，表示正则表达式flags是否有"g"；ignoreCase属性是一Boolean值，表示正则表达式flags是否有"i"；multiline属性是一Boolean值，表示正则表达式flags是否有"m"；lastIndex属性指定从何处开始下次匹配的一个字符串类型的位置索引，当需要时该值会转换为一个整型数。

RegExp.prototype.toString扩展改变了默认的toString方法，当正则表达式需要应用toString方法进行字符串转换的时候，返回的就是正则表达式的source属性值。

例如：/^\.dd\d+$/ + ""的结果就是"^\.dd\d+$".  


于是，/3/-/2/实际上等同于"3"-"2"，因此结果就是1.

如果没有toString方法的重置，/3/-/2/实际上等同于"/3/"-"/2/"，因此结果就是NaN.

1. {break;4;} //返回的是？

结果是报如下错误：SyntaxError: unlabeled break must be inside loop or switch.

意思是——解析错误：未标记的break必须在循环或switch中。

对照错误，我们加个标记，使之成为标记语句，就不会出错了。类似下面：

foo: { break foo; 4;}

1. 'foo' == new function(){ return String('foo'); }; //返回的是？

结果为：false.

由于这里是==, 'foo'又是正宗的字符串，因此，后面的new...需要转换成字符串。

new function(){ return String('foo'); } + ""; // "[object Object]"

显然，'foo' == "[object Object]"为false.

本题如果稍作一点修改，则结果完全不一样：

'foo' == new function(){ return new String('foo'); }; //返回的是？

结果为：true.

为何？

在怿飞（圆心）“[详解new function(){}和function(){}()](http://www.planabc.net/2008/02/20/javascript_new_function/)”一文中如下的解释(//zxx: 08年初的文章啊，那个时候我处在被女神嫌弃，发奋图强的阶段。)：

只要new表达式之后的constructor返回（return）一个引用对象（数组，对象，函数等），都将覆盖new创建的匿名对象，如果返回（return）一个原始类型（无return时其实为return原始类型undefined），那么就返回new创建的匿名对象。

什么意思呢？众所周知，JavaScript中有5种基本类型（Undefined类型、Null类型、Boolean类型、Number类型、String类型），如果new后面的function return的是这5中基本类型之一，new会认为你是纯屌丝，不理你，还是返回自己创建的匿名对象；当然，如果返回数组啊、函数、对象这类高富帅，new立马变龟孙子了，返回的就是这些高富帅。

由于String("foo")是字符串，而new String("foo")是对象。因此，前者返回的是匿名函数对象——显然不等于"foo"；后者就是new String("foo")对象，加上"foo" == new String("foo")，于是，结果为true.

1. 'foo'.split('') + [] //返回的是？

结果是："f,o,o".

记住，数组加数组，字符成老母。'foo'.split('')结果为数组["f", "o", "o"]，其变身字符串就是"f,o,o"跟后面的[]也就是""相加，就是最终的"f,o,o"了。

下面考考你，

[1, 2] + [3, 4] //返回的是？

是不是"1,2,3,4"？恭喜你，http://mat1.gtimg.com/www/mb/images/face/99.gif，回答…………错误！http://mat1.gtimg.com/www/mb/images/face/64.gif

这又是整哪样啊！哥，你只是稍微粗心了点。[1, 2]变成字符串是"1,2", [3, 4]变成字符串是"3,4"，因此两者相加是"1,23,4"而不是"1,2,3,4".

空数组实际上是个很有意思的东西。

[] == 0 //true

!0 // true

![] // false

纯属题外东西，就不扩展了。

### 三、公子来时雪花飘，公子离去知了叫

不容易啊，终于看到结尾了，从五一前写到五一后。

正如开始提到了，纯属个人理解，虽观点基本都多方考证，难免还有不准确的地方，欢迎有相关研究的同行指正。

油枯灯尽，想不出什么油麦的话语了，就这样吧。我个人是学到很多东西，希望对您的学习也能有所帮助。

文章的一些提问欢迎回答，您的回答会对其他过来学习的人提供很多的帮助，人的价值不正是在于留下什么吗？http://mat1.gtimg.com/www/mb/images/face/14.gif

末了，附上ECMAScript5.1中文版链接：<http://ecmascript.cn/>