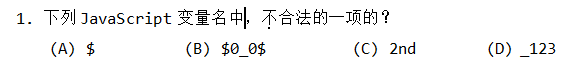
# JavaScript基础试卷讲解

# 第一题 标识符的命名规范

C



JavaScript中标识符：只能是英语字母、数字、下划线、美元符号组成。不能以数字开头。

var \_123;

var a;

var 123;

var $;

var \_;

不合法的通常考察数字开头的情况、怪异符号、关键字和保留字。

var 2a;

var a@b;

var a b;

var a+b;

var function;

var var;

var class;

拓展一个题目，下面不合法的JSON的键名是：

(A) $ (B)$0\_0$ (C)2nd (D)\_123

答案是没有答案，全都合法。**JSON的键名是任何字符串都可以当做键名的，但是如果键名不符合标识符的命名规则，则必须加上引号，访问属性的时候只能用方括号语法，不能直接打点调用**。

<script type="text/javascript">

var obj = {

$ : 1,

$0\_0$ : 2,

"2nd" : 3, **// 这个属性键是2nd，不符合标识符命名规则，必须用引包包裹**

\_123 : 4

}

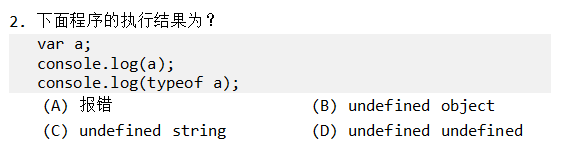
//alert(obj.2nd); **//这个代码错误！**

alert(obj["2nd"]); **//必须用方括号的语法**

</script>

# 第二题 关于undefined

D



我们说用var可以定义（或者叫做声明，definiation、declaration）一个变量。如果一个变量仅仅定义了，没有赋初值，此时这个变量的值是undefined。

这是这个语言好玩儿的地方，我明明用var去定义你了，但是你还是undefined（中文意思：没有定义）。

我们使用一个没有定义的变量：

<script type="text/javascript">

alert(a);

</script>

报错：



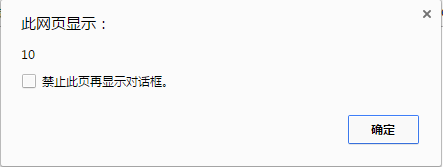
如果a我们不用var去定义，而是直接赋值，我们发现a就能弹出：

<script type="text/javascript">

a = 10;

alert(a);

</script>



也就是说JS中赋值这个行为，比var来得重要的多。

我们感觉不写var挺好，为什么要写呢？**因为JS中有一个规定，当一个变量没有var过而直接赋值的时候，系统会在全局作用域下自动帮你var，此时这个变量将是全局变量，可能不符合你的预期**：

<script type="text/javascript">

function fun(){

a = 100; //这个a是个局部变量么？不是，它是一个全局变量。因为没有写var。

}

fun();

alert(a);

</script>

如果一个变量仅仅var了，但是没有赋初值，此时它的值是undefined：

<script type="text/javascript">

var a;

alert(a); //undefined

alert(typeof a); //undefined

</script>

特别注意：

1. undefined的来源只能是这一个，就是变量var了，没有赋值。别的来源不会产生。
2. undefined是一个特殊值，类型也是undefined。undefined类型的值只有一个，就是undefined。
3. 我们在编程的时候，没有任何理由，没有任何立场去主动给一个变量赋值为undefined。但是不代表我们不用这个undefined这个值，通常用来检测变量是否被传入进函数：

function fun(a,b){

if(b == undefined){

alert("你需要传入两个参数！而你没有传入第二个参数！");

}

}

有工作经验的人会写：

function fun(a,b){

if(**!b**){

alert("你需要传入两个参数！而你没有传入第二个参数！");

}

}

# 第三题 NaN和Infinity

NaN表示not a number，中文意思不是一个数字。

Infinity表示无穷大，有正负之分，-Inifinity。

它们都是number类型的值。也就是说，NaN不是类型，是值。Inifinity不是类型，是值。

<script type="text/javascript">

alert(typeof NaN); //number

alert(typeof Infinity); //number

alert(typeof -Infinity); //number

</script>

先说NaN的哲学：**一切数学运算、数学函数需要产生一个数字结果，但是臣妾真的产生不出来的时候，会得到NaN**。

比如减法运算，结果一定是number类型的值：

8 - 3 //5

但是如果你：

"我" - "你" //?

减法运算的结果一定是数字类型，但是得什么数感觉差点意思。此时产生NaN。

再比如parseInt()函数结果一定返回数字，但是如果一个字符串（不是数字的字符串）此时返回NaN：

var a = parseInt("我");

alert(a);

数学运算的结果，如果不好产生值，结果就是NaN：

alert("我" % "你");

alert("我" \* "你");

alert("我" / "你");

alert("我" - "你");

alert(Math.sqrt("我"));

说说无穷大，记住：

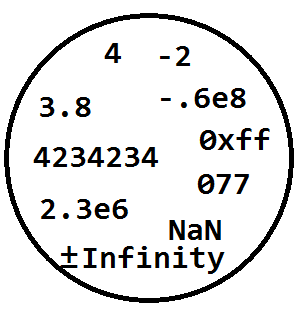
1. 0除以0等于NaN
2. 非0除以0等于Infinity或-Inifity

alert(0/0); //NaN

alert(5/0); //Infinity

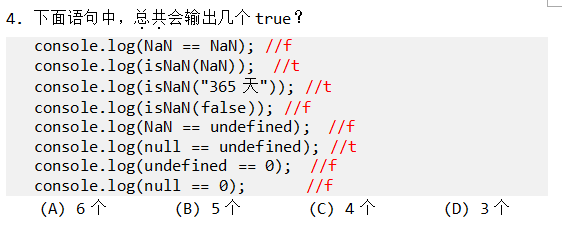
alert(-5/0); //-Infinity

也就是说，number类型的值：



# 第四题 关于isNaN函数

D



JavaScript是个怎么样的语言呢？千疮百孔的语言。在1996年JS被设计的时候，就缺乏顶层的思考。导致语言核心部分，很乱，面试的时候都选择比较乱的东西来考察面试者。

NaN不能自等：

<script type="text/javascript">

alert(NaN == NaN); //alert

</script>

这是一个没有道理的设定。

其他特殊值，都是可以自等的：

console.log(Infinity == Infinity); //true

console.log(undefined == undefined); //true

console.log(null == null); //true

此时产生了问题，NaN如何证明自己是NaN，JS发明了一个内置函数，isNaN()

alert(isNaN(NaN)); //true

这个函数发明还不如不发明，此时让世界变得更乱了！因为**不仅仅是NaN在inNaN()中会返回true，其他的一些情形在isNaN中也会返回true**，所以NaN还是无法自证。

我们跑一个题，先说说一个内置的构造函数，叫做Number()，注意N大写了。这个是一个内置的类，是一个内置的构造函数，至于类是什么、构造函数是什么，黄老师告诉你们，但是现在我们仅仅使用它。

**Number()函数是用来将其他类型的值，隐式转为数字类型的函数**。

console.log(Number("8")); //8

console.log(Number(true)); //1

console.log(Number(false)); //0

**console.log(Number(undefined)); //NaN**

console.log(Number(null)); //0

**console.log(Number("我")); //NaN**

console.log(Number("2017")); //2017

**console.log(Number("2017年")); //NaN**

**isNaN()函数内部在偷偷运行Number函数，如果Number函数返回的值是NaN，则isNaN()将返回true**。

console.log(isNaN("8")); //

console.log(isNaN(true)); //

console.log(isNaN(false)); //

**console.log(isNaN(undefined)); //true**

console.log(isNaN(null)); //

**console.log(isNaN("我")); //true**

console.log(isNaN("2017")); //

**console.log(isNaN("2017年")); //true**

isNaN()函数将所有不能隐式转换为number类型的值，都会返回true。NaN本身也是true。

特殊值的等等比较，没有道理可言，必须死记！

NaN == undefined; **//false**，NaN连自己都不等，凭什么等于你。

null == undefined; **//true**

undefined == 0; **//false**，因为undefined隐式转为NaN，所以不会等于0

null == 0; **//false**，这是一个特例，Number(null)结果真的是0，但是JS是千疮百孔的，很多东西不讲道理，这里null和0作比较返回false。也可以理解为null是一个特殊对象，0是一个基本类型值，不能相等。

# 第五题 连字符

+两边不都是number类型的时候，将做连字运算。

var a = "2.2";

var b = 1.1;

var c = parseInt(a + b); //parseInt("2.21.1");

console.log(c);

结果是2

# 第六题 变量声明的提升

**JS在执行程序时候，会先预览整个程序**。回忆一下，你高考的时候，考语文的时候，先去看最后的作文题目。

预览程序的时候，会提升变量的声明，即使你在使用变量之后你定义的变量，此时JS会自动的提升这个变量。

下面的程序会报错：

<script type="text/javascript">

alert(a);

</script>



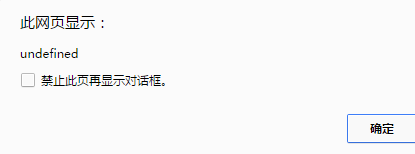
下面的程序不会报错，会弹出undefined：

<script type="text/javascript">

alert(a);

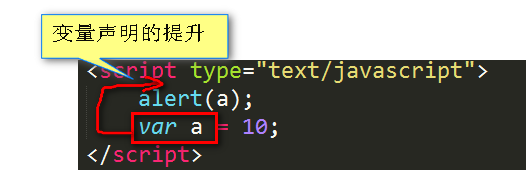
var a = 10;

</script>



JS在预览程序的时候，会先把var给你提升，JS心里想，哦，这个东西被定义了。但是不会提升赋值。

**变量的提升仅仅能提升声明，但是不会提升赋值**！



如果一个函数中有var，此时这个变量的生命头将自动提升在函数内第一行语句之前：

这是一道特别经典的JS面试题目：

<script type="text/javascript">

var a = 10;

function fun(){

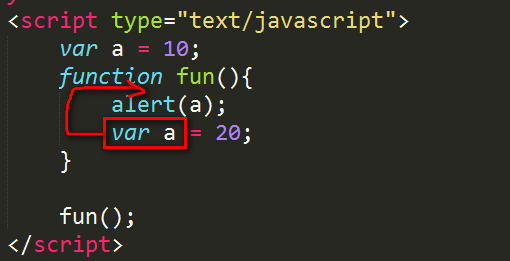
alert(a); **//undefined**

var a = 20; //此时因为JS程序预解析，这里的var a提升到了函数里面第一句话，见下图

}

fun();

</script>



提升了var a但是没有提升那个20，所以就相当于仅仅var了没有赋初值，我们说等于undefined。

注意两个事情：

**1. *JS*代码预解析阶段，对JS变量的提升是无条件的，无视if语句和for循环、while等语句。**

<script type="text/javascript">

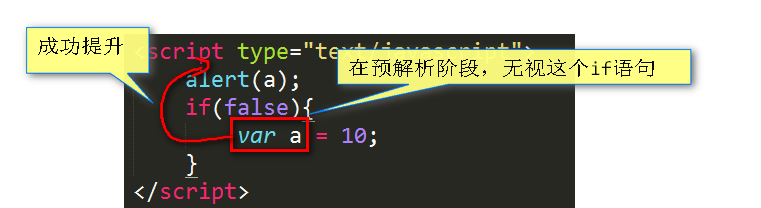
alert(a); //undefined

if(false){

var a = 10;

}

</script>



我们的考试题目：

<script type="text/javascript">

var foo = true;

function bar(){

if(!foo){

var foo = 10;

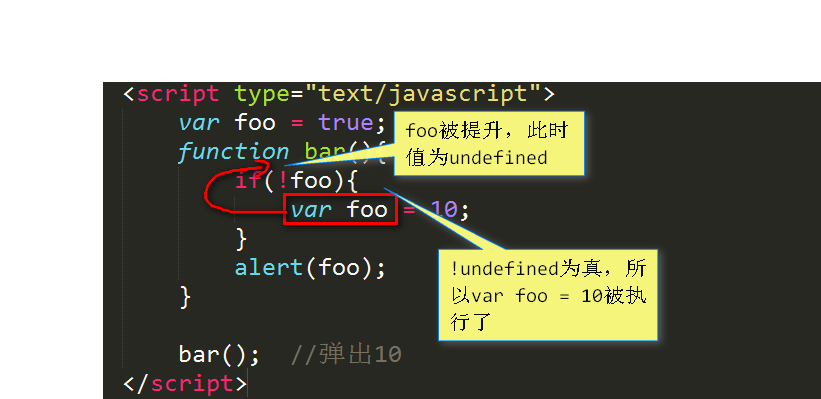
}

alert(foo);

}

bar(); //弹出10

</script>



再来一题：

var a = true;

function fun(){

if(a == undefined){

var a = 10;

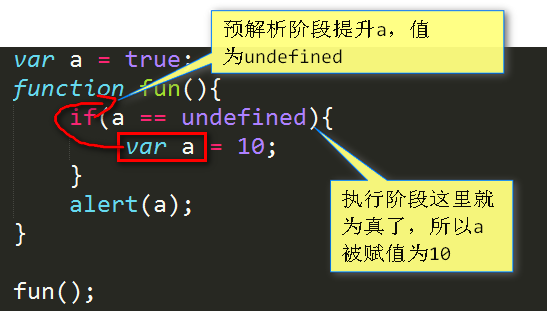
}

alert(a);

}

fun();

答案是弹出10。因为：



再来一个，一定要记住var的提升是无条件的，无视if、for、do、while等等语句：

alert(m);

for(var i = 200 ; i <= 100 ; i++){

while(false){

if(false){

do{

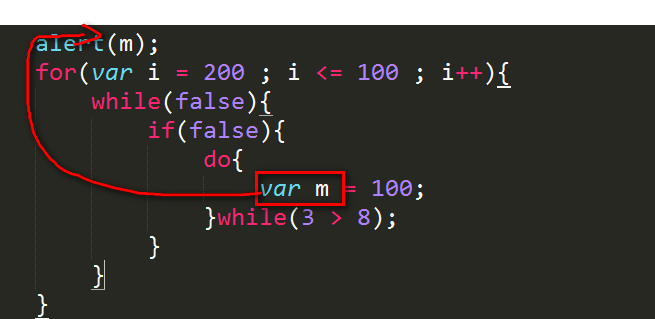
var m = 100;

}while(3 > 8);

}

}

}



**2.如果函数里面有实参，则函数体内的同名变量将不再提升**

下面的案例弹出8：

function fun(a){

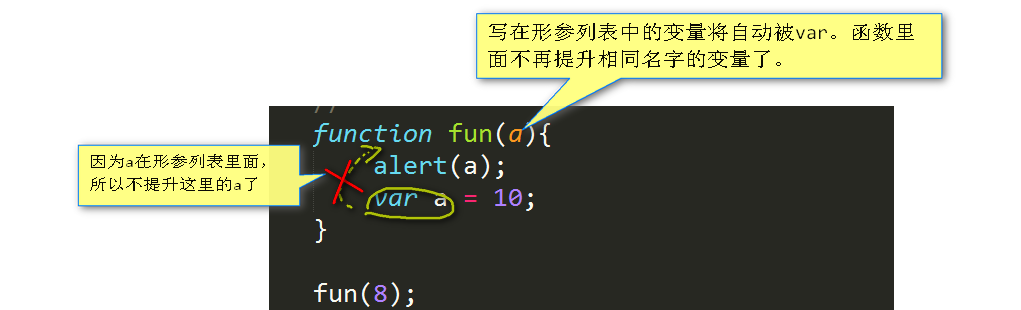
alert(a);

var a = 10;

}

fun(8);

原因：



# 第7题 自增运算符

a++ 先用再加

++a 先加再用

<script type="text/javascript">

var a = 3;

var b = 4;

var c = a++ + b++ + ++a + ++b;

console.log(c);

</script>

答案是18，原式 = 3 + 4 + 5 + 6 = 18。

不要认为a++是正确写法，工作中不用++a，错！！++a的写法非常常见！比如：

当所有图片加载完毕之后，弹出alert框：

<script type="text/javascript">

var imgs = document.getElementsByTagName("img");

var count = 0; //已经加载完毕的图片数量

for(var i = 0 ; i < imgs.length ; i++){

imgs[i].onload = function(){

//某张图片加载完毕之后做的事情

**++count** == imgs.length && alert("所有图片已经加载完毕！");

}

}

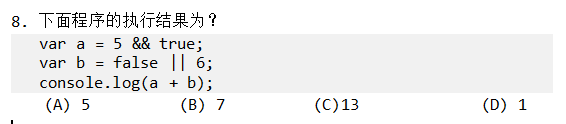
</script>

# 第8题 短路语法

短路总结，不建议你去背诵，和打快板

**a && b , a真抛b，a假抛a**

**a || b ， a真抛a，a假抛b**



且运算，5是真了，所以&&运算符后面的值决定总值，所以抛出true，a的值是true。

或运算，false是假了，所以万众瞩目后面，抛出后面。b是6。

原式 = true + 6 = 7

因为在做数学运算的时候，true变为1。

实战中，我们这样使用&&短路：

**++count == imgs.length && alert("所有图片已经加载完毕！");**

实战中，我们这样使用||短路：

function fun(age){

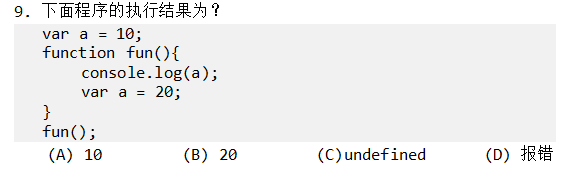
**var age = age || 18;**

}

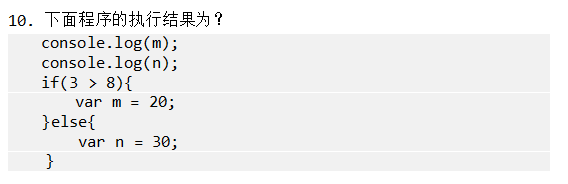
面试是一码事儿，实战又是一码事儿。

# 第9题 变量声明提升

C

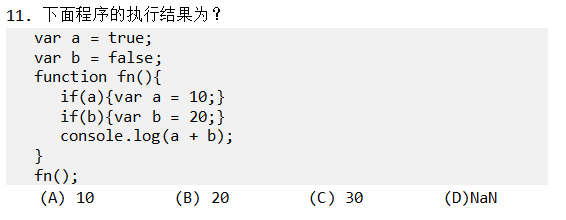


# 第10题 变量声明提升

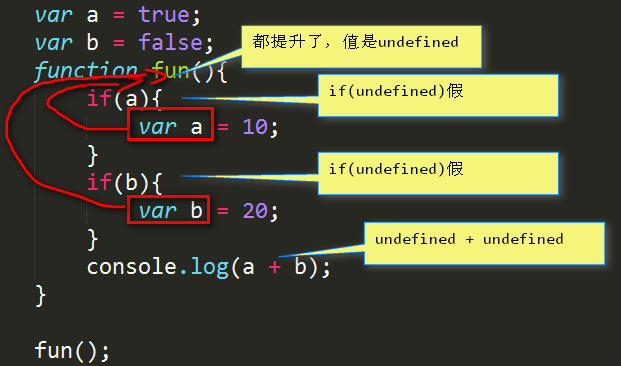


undefined undefined ， 因为无视if，都会提升

# 第11题 变量声明提升



别看外面a的值、b的值，在函数里面都会提升为undefined。所以就是undefined+undefined = NaN。



undefined参与任何运算都等于NaN。

# 第12题 函数声明提升优先于变量声明提升

这个点也是面试爱考察的点，

报错，因为var定义的函数，不能提升的：

<script type="text/javascript">

fun();

var fun = function(){

alert("我是fun函数");

}

</script>

能够提升:

<script type="text/javascript">

fun();

function fun(){

alert("我是fun函数");

}

</script>

如果名字冲突，函数优先变量提升：

<script type="text/javascript">

console.log(fun); **//函数**

var fun = "B";

console.log(fun); **//B**

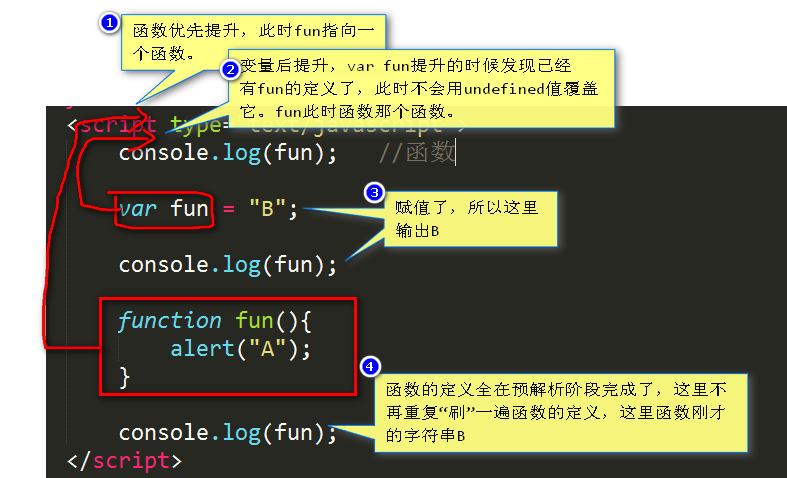
function fun(){

alert("A");

}

console.log(fun); **//B**

</script>



来一个难题，这个题目做对了，没有问题了，答案DABBB：

<script type="text/javascript">

fun();

var fun = function(){

alert("A");

}

fun();

var fun = function(){

alert("B");

}

fun();

function fun(){

alert("C");

}

fun();

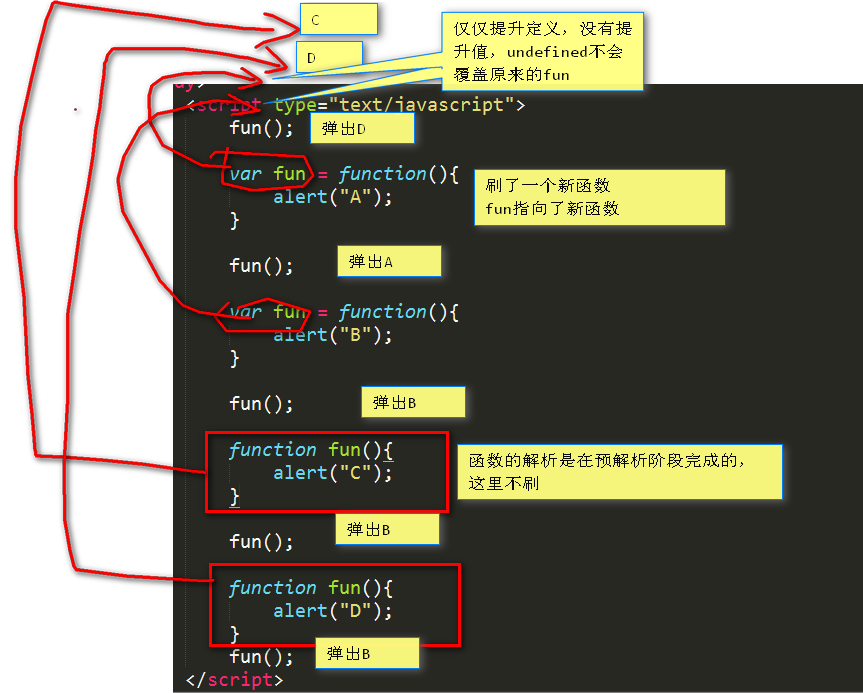
function fun(){

alert("D");

}

fun();

</script>



再来一题，不画图了，自己琢磨，答案DDBCC：

<script type="text/javascript">

fun();

function fun(){

alert("A");

}

fun();

var fun = function(){

alert("B");

}

fun();

var fun = function(){

alert("C");

}

fun();

function fun(){

alert("D");

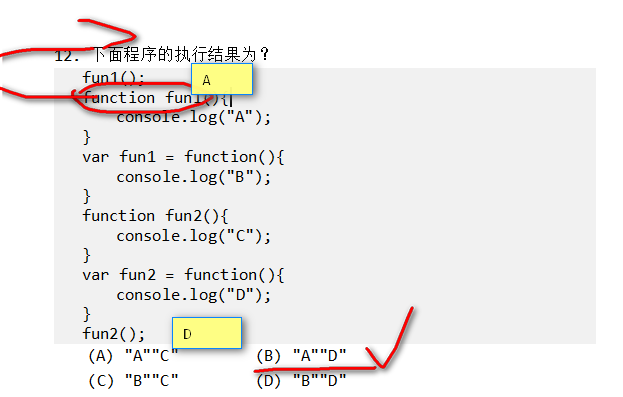
}

fun();

</script>

考试题目就迎刃而解了：

B



# 第13题 引用类型值

JS中有两种变量，或者叫做两种值：

* 基本类型值：number、string、boolean、undefined、null
* 引用类型值：Object、Array、Function、RegExp、Math、Window等等

区别：

1. 【传值和传址】基本类型值传给变量的时候，传入的是值，或者说是值的副本；引用类型值传给变量的时候，传入的是地址，指向的是同一个变量。

2. 【验值和验址】做==判断的时候，引用类型值是比较的是内存地址，要验证两个值是不是同一个内容地址，如果是返回true。

比如：

<script type="text/javascript">

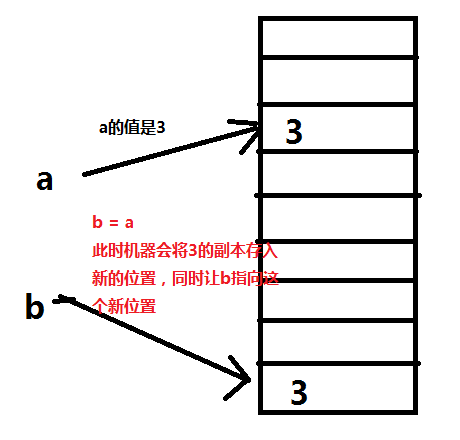
var a = 3;

b = a;

b = 9;

alert(a); //3

</script>



基本类型传入了变量，此时变量指向的是值的副本。

引用类型不同：

<script type="text/javascript">

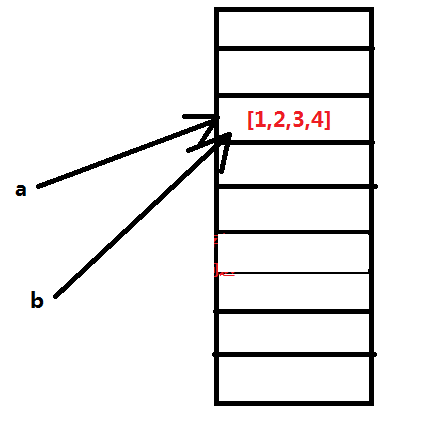
var a = [1,2,3,4];

b = a;

b.push(8);

console.log(a); //[1,2,3,4,8]

</script>



a、b指向了内存中的同一个地址，所以a变b也变。

再来一题：

var obj1 = {"a" : 1 , "b" : 2};

obj2 = obj1;

obj1.b++;

obj1.b++;

obj1.b++;

obj1.b++;

obj1.b++;

obj1.b++;

console.log(obj2); // {"a" : 1 , "b" : 8}

obj1和obj2指向的是内存中同一个地址。

再来一题：

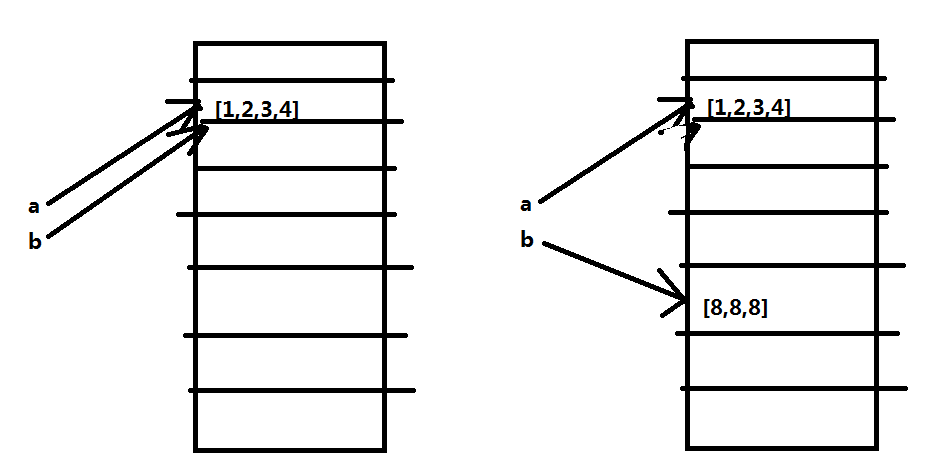
var a = [1,2,3,4];

var b = a;

b = [8,8,8];

console.log(a); //不受影响，仍然为[1,2,3,4]

当你使用新的字面量给一个变量赋值的时候，此时受影响的仅仅是这个值，和其他指向的变量无关！



注意引用类型值进行==或者===的验证的时候，判断的是他们是不是指向同一个内存地址：

alert([1,2,3] == [1,2,3]);

都是数组1、2、3，但是是内存中两个数组，此时返回false。

都是false：

alert([1,2,3] == [1,2,3]);

alert({} == {});

alert(/\d/ == /\d/);

数组、对象、正则表达式都是引用类型值，要验证地址是否相同。

都是true：

alert(Math == Math);

alert(window == window);

alert(document == document);

系统级别的内置对象，天生存在，并且仅仅存在一个，所以都是true。

# 第24题 引用类型值

考察引用类型值，答案是2

var o = {};

o.a = 1;

var b = o;

b.a = 2;

alert(o.a); //2

o和b都指向了同一个对象。

# 第25题 引用类型值

var arr1 = [4,5,6,7];

var arr2 = arr1; //arr1、arr2两个变量指向同一个对象

arr2[0] = 8; //[8,5,6,7]

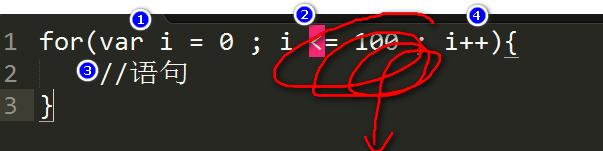
arr1.push(arr1.shift()); //[5,6,7,8] 数组删除头部的元素，将删除的头部元素在尾部插入

console.log(arr1); //[5,6,7,8]

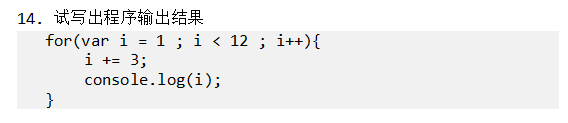
console.log(arr2); //[5,6,7,8]

# 第14题 循环语句

for循环的机理一定要清楚，语句的执行顺序要知道

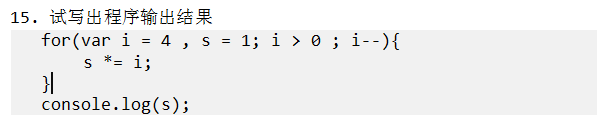


4、8、12



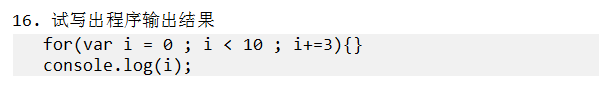
# 第15题 累乘器

算阶乘，s就是累乘器：



4的阶乘，4\*3\*2\*1 = 24

# 第16题 循环语句



循环体中没有内容，循环结束之后输出1个值。但是我们看看i的变化：0、3、6、9、12

循环结束之后，会输出12。

这个i退出循环之后，还有意义，在JS编程中很有用：

**请检查数组中[1,5,65,34,13,45,5,7,4,34]没有3的倍数。**

**换句话说就是验证数组中的数据都不是3的倍数。**

算法思想叫做筛子法，一个一个过验证，但是问题是，如何表述“都”是一个难题。可以妙用i的值出了for也有意义的特性，在for循坏结束之后验证i的值为多少。

<script type="text/javascript">

var arr = [1,5,65,34,13,46,5,7,4,34];

for(var i = 0 ; i < arr.length ; i++){

if(arr[i] % 3 == 0){

break;

}

}

**//如果正常结束了for循环，i的值此时是arr.length**

if(i == arr.length){

alert("数组中没有3的倍数！");

}else{

alert("数组中有3的倍数！");

}

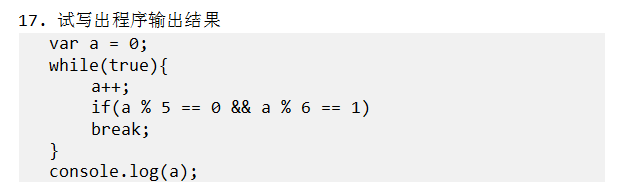
</script>

# 第17题 程序的宏观意义

千万不要傻傻的去读程序，而是应该去宏观的思考程序的意义。

这个程序在**寻找最小的能被5整除，被6除余1的正整数**。

我们就要看5、10、15、20、25、30、35……中谁除6余数是1，答案是25。

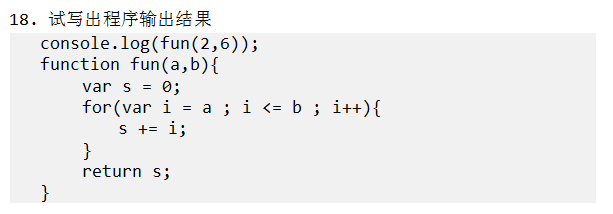


当你去寻找什么东西的时候，不知道范围是什么的时候、不知道终点是什么的时候，往往写while(true)死循环。一旦达到目的，终止循环。

# 第18题 程序的宏观意义

函数fun的功能是计算[a,b]区间内所有整数的和。

fun(2,6)显然等于2+3+4+5+6 = 20



# 第19题 变量的作用域

JS设计的非常简单，能够锁住、关住、限制住、控制住变量的作用域的，只有函数。

<script type="text/javascript">

**var m = 10;**

function fun(){

**var m = 20;**

m++; //局部变量

return m; //局部变量变为21

}

m += fun(); //外面的m10，在10的基础上加上了21，

console.log(m); //31

</script>

# 第20题 闭包

闭包非常简单，没有必要过分渲染它。

什么是闭包？闭包就是：

**当函数不在自己定义时的作用域运行的时候，能够记住自己定义的时候那个作用域**。

这个案例非常经典，无数人是通过这个案例认识闭包的，必须在15秒钟内写出来：

<script type="text/javascript">

function outer(){

var m = 100;

function inner(){

alert(m);

}

return inner;

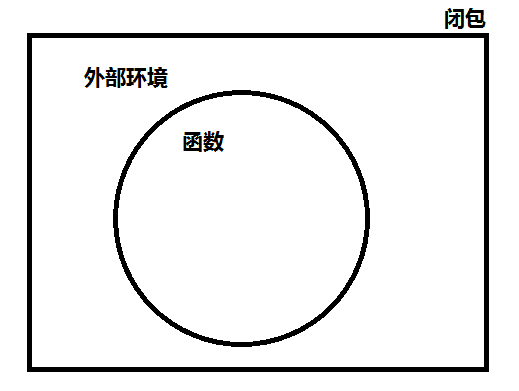
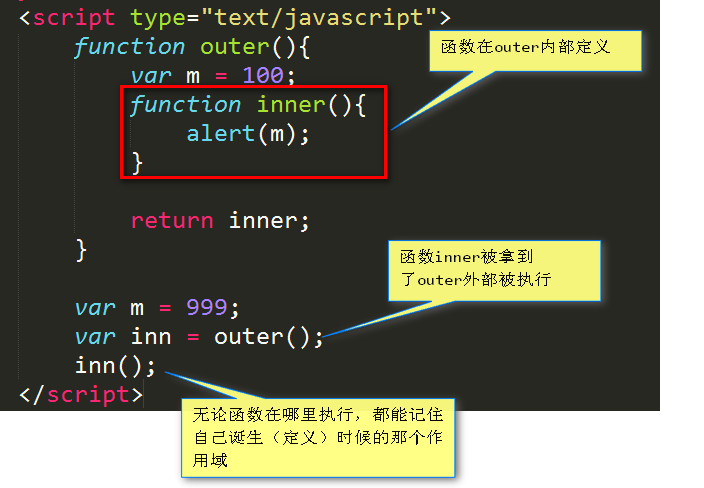
}

var m = 999;

var inn = outer();

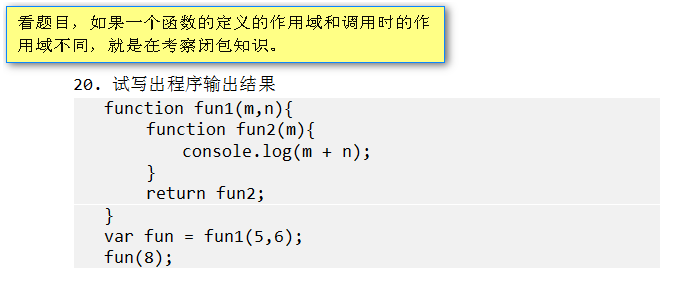
inn();

</script>



看题目，题目出的都特别好：

14



# 第21题 闭包

当同一个函数多次被返回的时候，此时得到全新的闭包。

先不说21题，先看：

function fun(){

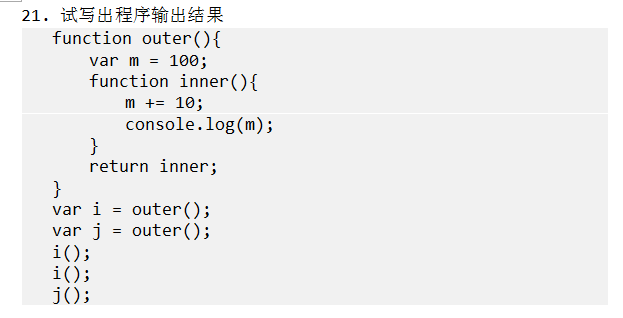
return "a";

}

var i = fun();

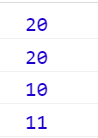
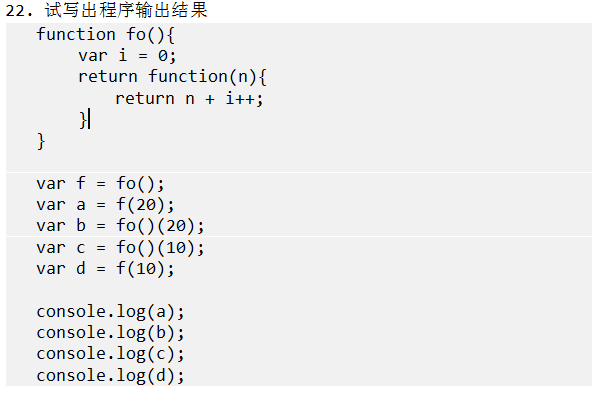
var j = fun();

console.log(i === j); //true，因为都是基本类型值，基本类型值比较值是否相同。



i和j是完全不同的两个函数，各自有各自的闭包。答案：110、120、110

# 第22题 闭包



//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function fo(){

var i = 0;

return function(n){

return n + i++;

}

}

var f = fo(); //f是个函数

var a = f(20); //f是函数，所以f就能运行。先用再加，a等于20，闭包里面的i已经变为1

var b = fo()(20); //外部函数fo又一次执行得到全新闭包，新闭包中的i不被刚才的f影响。此时仍是0，先用再加，此时是20。

var c = fo()(10); //同理，又是一个新闭包，等于10

var d = f(10); //老闭包，i已经是1了，此时是11

console.log(a);

console.log(b);

console.log(c);

console.log(d);

# 第23题 同步和异步

超前了，我们在jQuery中讲解animate的时候会介绍。这里稍微唠叨一下，后面课程再介绍:

JS是单线程的语言，运行时有两个队列：同步队列、异步队列。

* 同步语句就是for、if、函数调用等等；
* 异步语句就是setInterval()、setTimeout()、animate()、fs.readFile()、$.get()

异步语句的执行时机实在同步队列干净的时候才行。

所以来一个题目：

setTimeout(function(){

console.log("★");

},1);

for(var i = 0 ; i < 10000; i++){

console.log(i);

}

注意，★一定不会打断0~10000的输出，★一定出现在9999这个数字之后。

再来一题，这个题目中有两个队列，异步队列中依次是★和■。同步队列中是两边0~9999的输出：

setTimeout(function(){

console.log("★");

},1);

for(var i = 0 ; i < 10000; i++){

console.log(i);

}

setTimeout(function(){

console.log("■");

},1);

for(var i = 0 ; i < 10000; i++){

console.log(i);

}

答案是输出0~9999两边之后，输出★、■。

考试题目就迎刃而解了，即使延迟时间为0，也属于异步队列中的语句，此时要等到for()做干净才能执行，所以就是100个100。

for(var i = 0 ; i < 100 ; i++){

setTimeout(function(){

console.log(i);

},0);

}