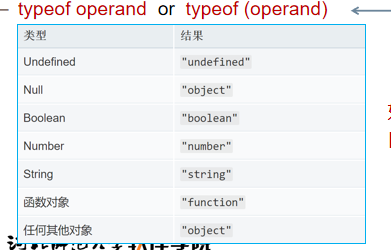
ES5

基本（原始）类型：Number,String.Boolean,Null,Undefined

引用（对象）类型：Object(Array,Function,Date,Error)

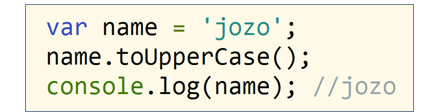
通过typeof操作符可以判断数据类型。·



JS中的变量是没有类型的，只有值才有。变量可以随时持有任何类型的值。这是动态类型。

数据内存中，栈和堆内存最大的区别是：栈内存是按值访问的，而堆内存是按引用访问的。

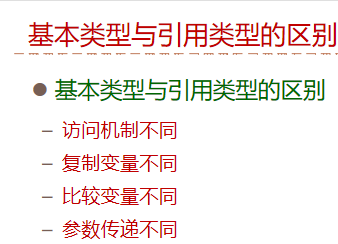
**基本类型的存储**：基本类型的变量是存放在栈区的，基本类型的值是不可变的。比如：



**toUpperCase() 方法用于把字符串转换为大写。一个新的字符串，在其中stringObject的所有小写字符全部被转换成了大写字符。**

然而这里的var name=’joze’在控制台上输出时依旧是小写。这就说明了基本类型的值是不可变的。

**引用类型的存储：**引用类型的值是同时保存在栈内存和堆内存中的对象。



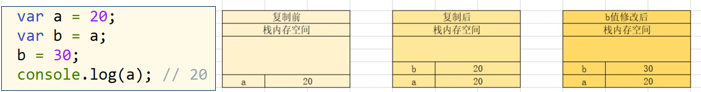
**访问机制：**

原始类型的值直接访问。（比如a=20就能直接在栈内存空间访问到。）

对象类型的值通过引用访问，不能直接访问。（比如a1=[1,2,3]就需要从栈内存空间中先获取a1的访问地址，然后到堆内存中根据地址获取值）

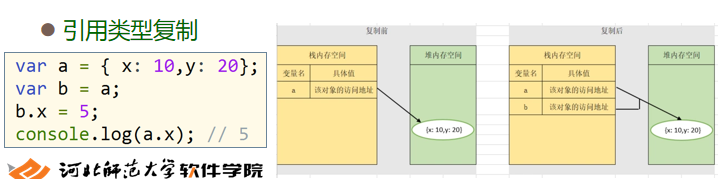
**复制变量：**

基本类型复制：相互独立互不影响。



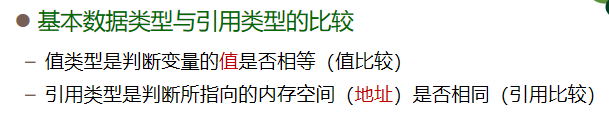
这是各自在栈内存里面存储了。相互不影响。

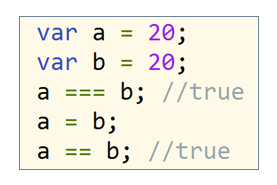
引用类型复制：



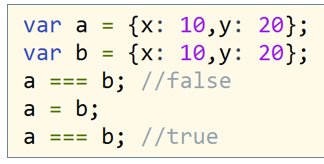
引用类型由于是根据栈内存里的访问地址找的，因此var b=a;之后，b跟a的地址就是一样的，这时他们就指向了同一片内存空间，这时b.x=5修改了值就会把内存空间的值就修改了，而a的访问地址依旧没变，依旧指向这片空间，但是这个时候x的值已经改变，因此输出的就是a.x=5；

比较变量：





这里 a=20;b=20;在栈内存中是各自存的，直接在栈内存中就可以找到他们的具体值。因此输出的都是true；



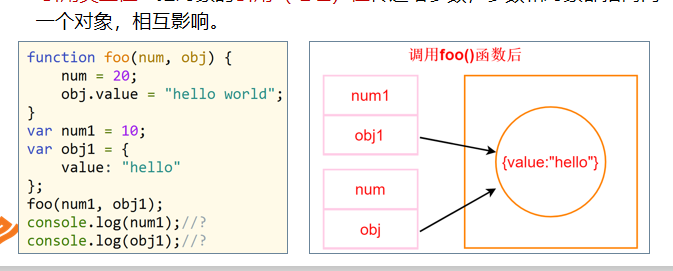
这里a={x:10,y:20};b={x:10,y:20};他们虽然括号内的数据是一样的，但是他们在堆内存中指向的空间是不一样的，因此他们是不相等的，但是a=b把b的访问地址赋值给了a，因此他们指向的是同一片空间，所以他们是相等的。

参数传递：

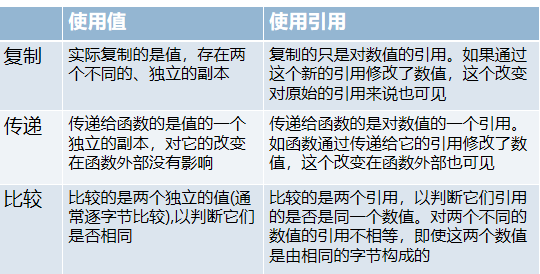
ECMAScript中所有函数的参数都是按值来传递。

基本类型值：把变量里数据值传递给参数，之后参数和变量互不影响。

引用类型值：把对象的引用（地址）值传递给参数，参数和对象都指向同一个对象，互相影响。



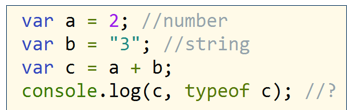
在这道题中，foo(num1,obj1);把实参带进函数执行。由于num=20;num1=10;是基本类型，因此值没有改变。Obj.value=”hello world”;这时的obj=obj1，因此obj跟obj1在没有执行Obj.value=”hello world”;之前，他们指向的是同一片堆内存空间，在执行Obj.value=”hello world”;之后，value改变了，因此输出了hello world



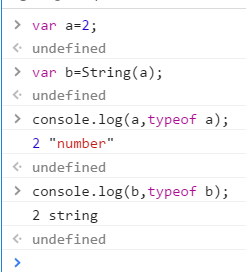
**类型转换：**将值从一种类型转换成另一种类型。

隐式类型转换：通常是某些操作的副作用，不易看出

显示类型转换：可以在代码中明显看出

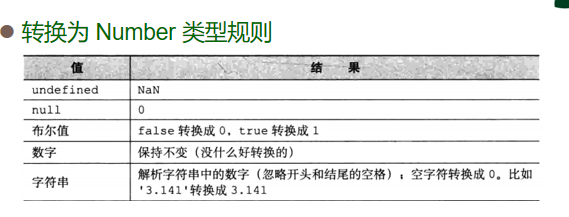


这题中，c=a+b，b是string，a是number，因此+是拼接，变成了“23”，发生了隐式类型转换。C=”23”，c是string类型



var b=String(a);就是强制把a（number类型）转换成string类型

强制转换成number类型的有：parseInt(),parseFloat(),number()



NaN:

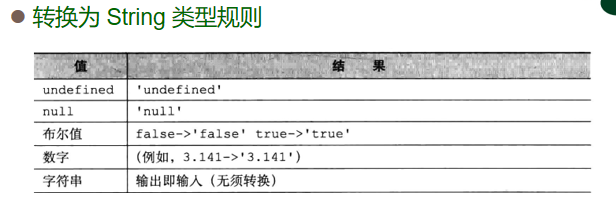
1. 便是一个没有意义，不正确的数值
2. Console.log(typeof NaN);-----Number
3. **与自身不相等**

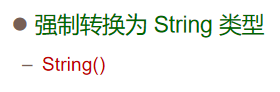
IsNaN()

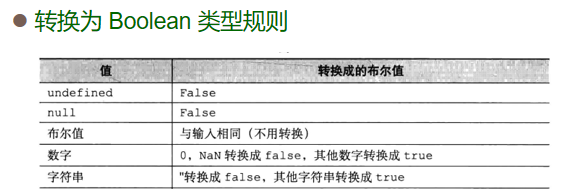
用来检测参数是否为NaN值（后面有一道题就有用到）

参数是”NaN”时返回true,否则返回false

IsNaN(“123abc”)----true









包装对象：

JS对象时一种复合值，是属性的集合

