

C：底层软件开发

C++：VR虚拟

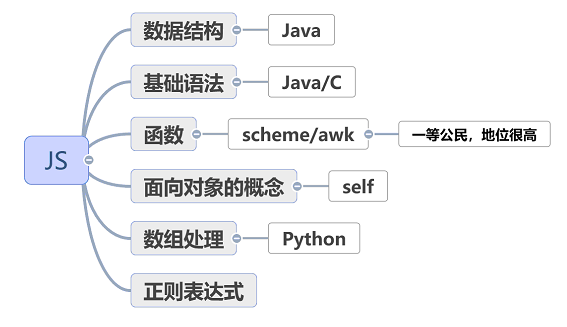
C#：窗体应用，客户端异步编程，游戏开发

Java：嵌入式 大型网站

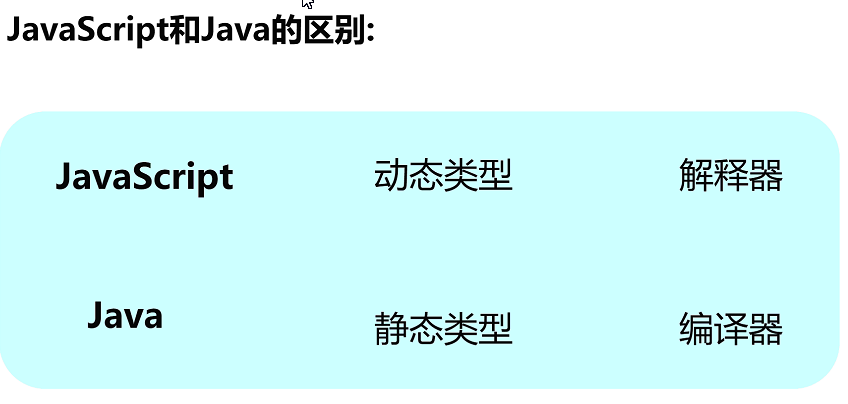
Ptyhon：科学计算

**JavaScript：前端逻辑的交互**

# Js的组成：



**区别：**



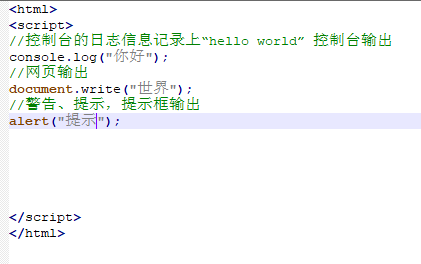
**动态语言：声明变量时*不需要*给定类型，在代码运行过程中会发生改变**

**静态语言：声明变量时*需要*给定类型，在代码运行过程中不会发生改变**

**解释器：代码运行中，每行依次运行**

**编译器：先整个代码全部翻译完成，再运行**

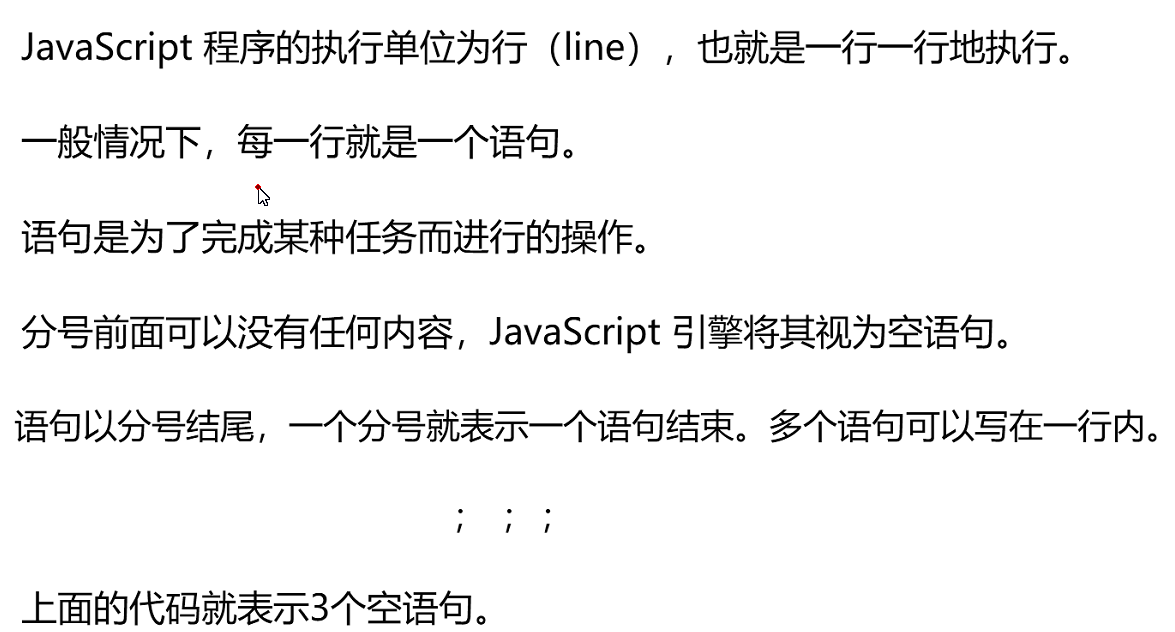


****

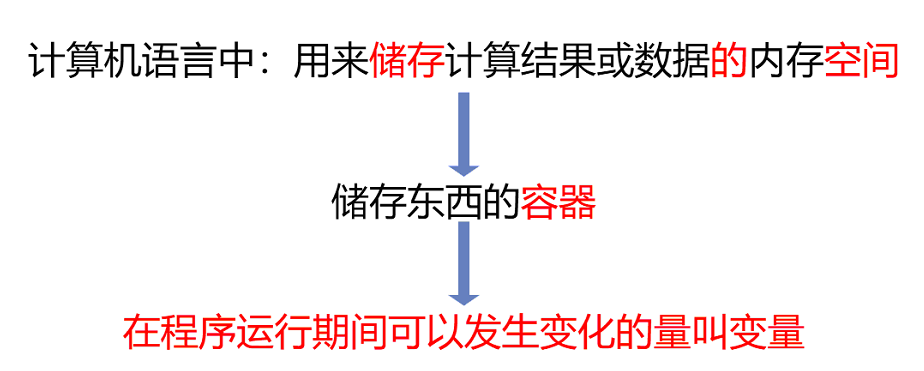
**Console.log(“”);控制台输出显示，主要作用于代码报错时能快速找到报错位置**

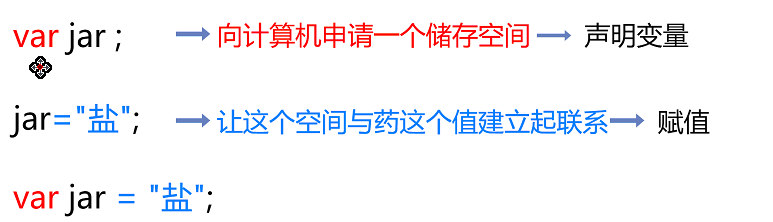
**document.write(“”);网页显示输出**

**alert(“”);网页弹窗输出显示**



**变量**



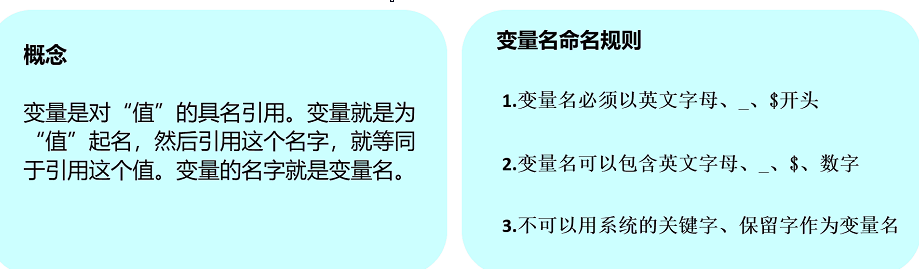


**Var jar =”盐”;**

**Jar：声明一个变量**

**Var：声明命令**

**名字相同的变量赋值不同的话，最后一个值会覆盖前面的值**



**变量名必须用英文字母、\_、$开头**

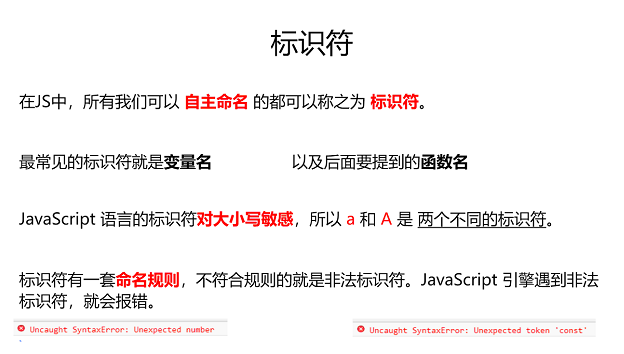
**变量名可以包含英文字母、\_、$开头**

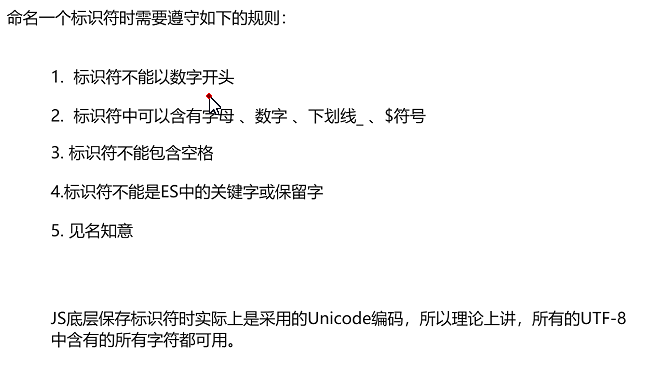
**不可以用系统的关键字、保留字作为变量名**

**JS区分大小写**

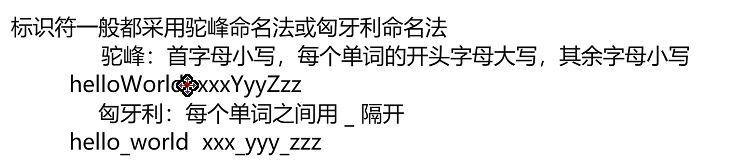
**A和a是两个不同的东西**

标识符：自主命名的都叫标识符





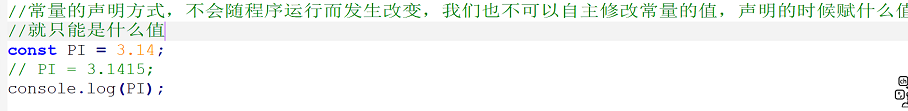
注：Number const均为特殊，不能引用为变量名



变量名特殊符号只能用\_和$，不能用数字开头（但可以在后）

**常量**

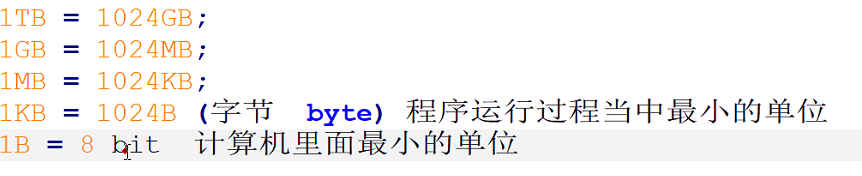
Const声明常量，用法与变量相似。

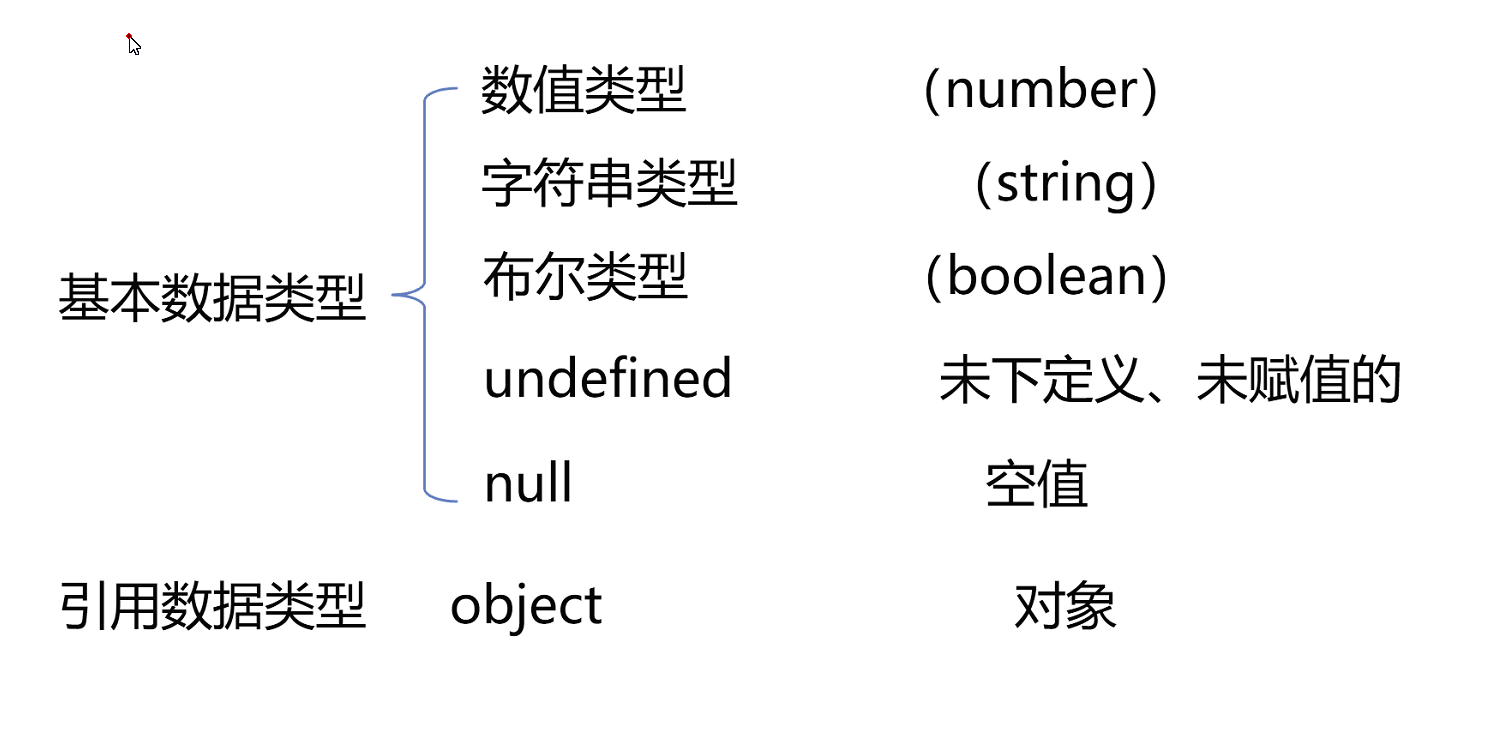


声明变量时常用小写字母，声明常量时常用大写字母

常量声明是什么值，就只能是什么值，无法被动或者主动修改。

**数据类型**





Number：数值类型包含整数和浮点数

以64（bit）位浮点数储存。（最高精确17位，小数点后最多17位或者小数点前最多17位）

JS里没有整数，都是以小数存在

最高精度17位 数出来的

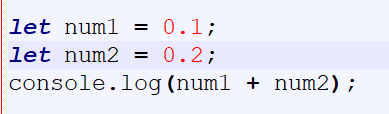
64位 64个0或者1

1. （0表示正数，1表示负数）

64位二进制 从左往右的第一个

2——12表示的整数部分，决定数值的大小

13——64 表示的是小数部分，决定数值的精度



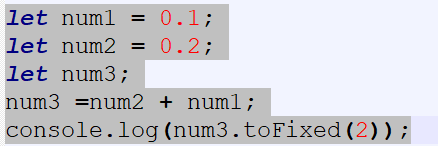
结果：



原因：js 在储存浮点数的时候会有一个摄入误差，并不是所有的浮点数转为二进制都是精确地

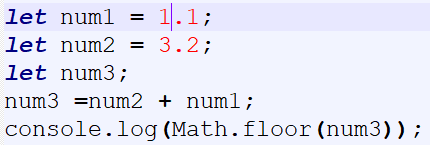
数值常见的处理：

保留多少位小数 toFixed(); 括号里填的数值表示保留多少位小数

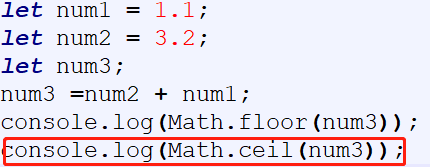


取整的方式：

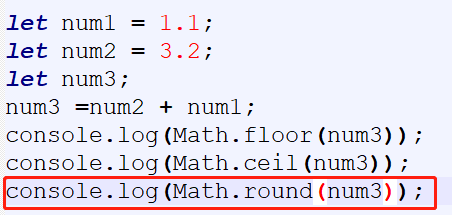
Math.floor(); 向下取整 只保留整数部分，小数部分掐掉



Math.ceil(); 向上取整 只保留整数，不管小数部分值多少 直接往整数进一

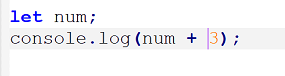
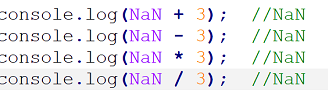


Math.round(); 四舍五入取整



特殊数值1：

NaN not a number (不是一个数字)

1表示不是一个数字，但是检测之后又是数值类型，

2参与任何运算得到的结果都为NaN。

3计算错误的情况下会出现NaN

4 NaN与任何数都不相等，包括它自己

如何检测变量的值是一个NaN:

console.log(isNaN());





确定 不是一个数字吗？

Console.log (NaN - NaN); //NaN

特殊数值2：

**Infinity 表示无穷大 -Infinity 表示无穷小**

**除法操作，除数为0时得到Infinity**

Console.log (5 / 2);2.5

Console.log(0/2);0

Console.log(5 / 0); Infinity

Js数值的取值范围：

Number.MIN\_VALUE——Number.MAX\_VALUE

最小范围 最大范围

检测数据类型的方式：

Typeof（）；

用法：console.log(typeof （num1))

检测num1的类型

复杂数据类型（引用数据类型） object

关于var

Var 关键字 会造成变量的提升,赋值不会提升，整个程序不会报错，但后期发生意料之外的情况，难以找到bug

因此建议以后用let关键字替换var

Let关键字不会造成变量的提升

Let用法与var相同

# 字符串类型

String 一行、一串

账号： 密码：

字符串的写法：

1. Let name =“张三”;
2. Let name = ‘李四’；

要么都是单引号，要么都是双引号，引号成对出现，单引号和双引号不做区分，但不能共用

如果我们要在字符串当中直接输出“”或者‘’时，可以采用 内双外单 或者内单外双 这时候才能把字符串里面的引号看做是字符串的内容//或者使用for: \ 反斜杠

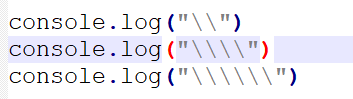
\t 水平制表符 前面缩进

\n 换行符

\” \’ 一个斜杠转义一个斜杠





\

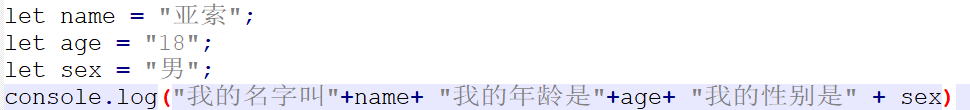
\\

\\\

模板字符串：

Console.log(` ${ }`);

姓名 年龄 性别



布尔值类型：boolean

值只有两个 true false

Undefined 未下定义的 未赋值的

表示的是一片可以访问，但是数据未知的空间

整个类型只有他本身undefined

出现的条件：

1声明变量，但是未赋值时

2调用函数，但是没有与形参相匹配的实参

3访问对象里不存在的属性

4函数的默认返回值

Null 空值

表示的是什么都没有，大部分情况用于占位置，定义为空值 已经定义了值

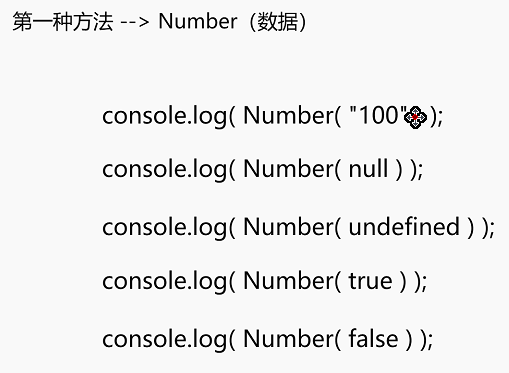
Null 表示“没有对象”，即此处不应该有值

Underdined 表示“缺少值”，即此处应该有一个值，但是还没定义

Object 引用数据类型，复杂数据类型

数据类型的转换

1其他类型转换为数值类型：



**Number（需要转换的数据）**

**转换规则：**

**如果字符串中存在数字和小数以外的其他字符，则转换失败，结果为NaN**

**开头有空白字符过滤**

**中间有空白字符则转换失败结果为NaN**

**加减号开头表示正负号**

**空字符串或空白字符串转换结果为0**

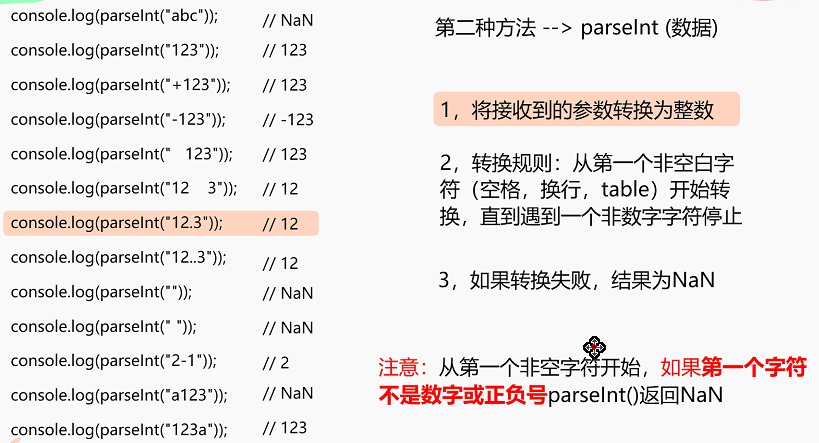


第二种方法，parselnt(数据)

**转换规则：**

1. **将接受到的参数转换成整数**
2. **从第一个非空白字符（空格，换行，table）开始转换，直到遇到一个非字符停止**
3. **如果转换失败，则结果为NaN**

**注意：如果第一个字符不是数字或正负号，则转换失败结果NaN**



其他类型转换为字符串类型

String（需要转换的数据）；

转换规则：把传进去的参数都添加一个双引号“”

第二种方法：（动态）

.toString();

注意的点：

通过我们声明出来的变量，使用.toString()的方法完成转换

使用这种方法，不能转换的有 null、underfined 其他和第一种方法一样

第三种方法：

转换规则：直接操作需要转换的数据 运用+操作 加上一个空字符串，其余规则和第一种方式一样

console.log(undefined + '');

Console.log(123 + “”);

**总结**

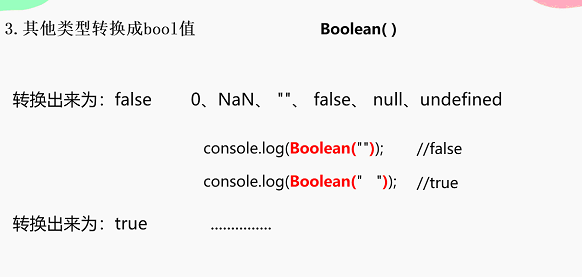


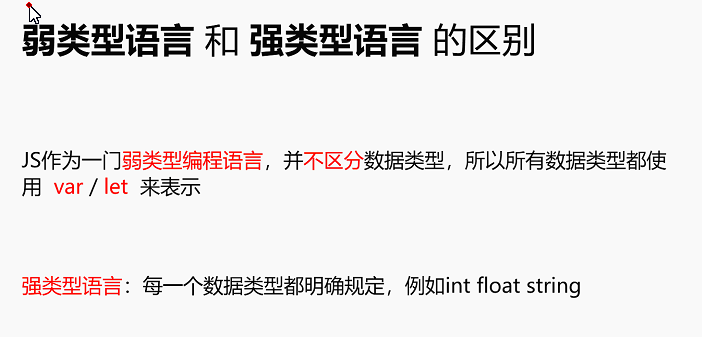
3其他类型转换为Boolean

Boolean（需要转换的数据）；

转换出来为false的：0、 “”、NaN、false、null、undefined、 !true

转换出来为true的：除了转换为false的都为true





**运算符**

**定义：告诉编译器执行特定的数学或逻辑操作的符号**

**Js里的运算符包换有：**

**算术运算符：**

**赋值运算符**

**自增自减运算符**

**关系运算符**

**逻辑运算符**

**三目运算符**

**位运算符**

**算术运算符：**

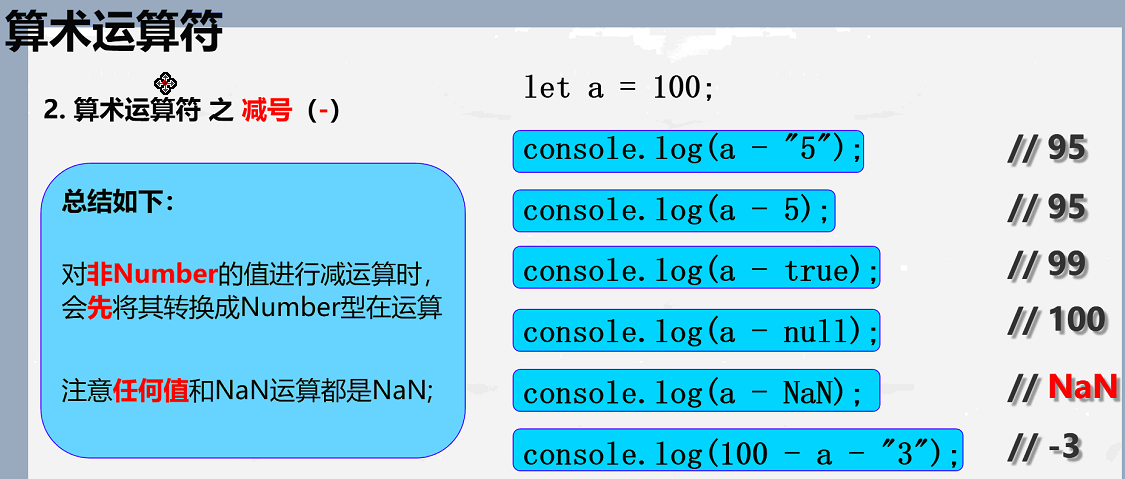
**常见的：+-\*/ %**

**加法运算：**

**对非Number类型（字符串除外）的值，进行加法运算会将其转为数值类型再计算**

**任何值与NaN运算，都是NaN**

**对于含有字符串类型的相加运算时，实际操作的是字符串的拼接，且优先级从左往右**

**减法**

**对非Number类型的值，进行减法运算会将其转为数值类型再计算**

**任何值与NaN运算，都是NaN**

**乘除和减法一样。**

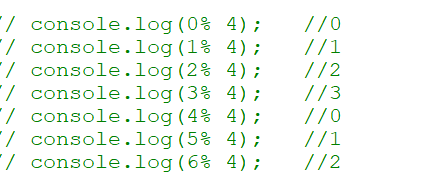
**取余（%） 取模：**

**余数的符号取决于被除数的符号，取余运算的结果必须以第一个操作的数的符号相同**

**作用：**

**1，50%的情况下用于判断一个数能否被另一个数整除,**

**2，用于控制一个变量的值在某一个范围内变换**



**// 商\*除数+余数=被除数**

**// 被除数-商\*除数=余数**

**乘方 开方：**

**乘方：**

**方法一**

**示例：2的3次方**

**Console.log(2 \*\* 3);**

**Console.log(2 \*\* -3);**

**方法二**

**Math.pow(x,y) 返回x的y次幂**

console.log(Math.pow(2,3)); //表示2的3次方

**开方：**

**方法一**

**Math.sqrt(); //直接开平方**

**console.log(Math.sqrt(25));**

**方法二**

**console.log(Math.pow(x,1/y)); //x的y次方**

**console.log(8 \*\*(1/3)); //8的3次方**

取整的方法：

Math.round() // 四舍五入取整

Math.ceil() //向上取整

Math.floor() //向下取整

parseInt() //截断小数取整

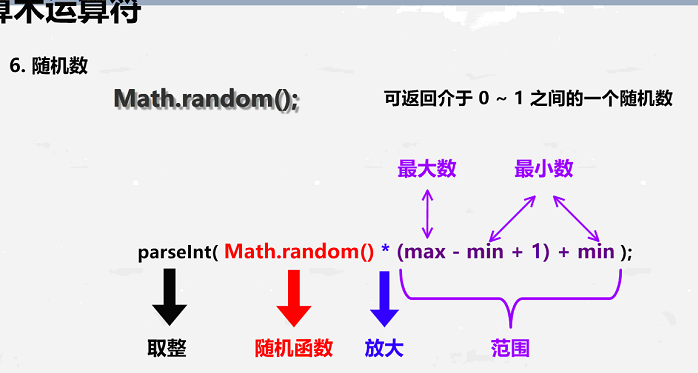
|0; //截断小数取整（数据过大 会发生数据截断）

**随机数**

**取范围随机数公式：**

**console.log(Math.random()\*((MAX-MIN)+1)+MIN |0);**

**取值范围：max --- min，包括max和min**



**赋值运算符**

**= let a = 10;**

**+= a = a + 5**

**-= a = a - 5**

**\*= a = a \* 5**

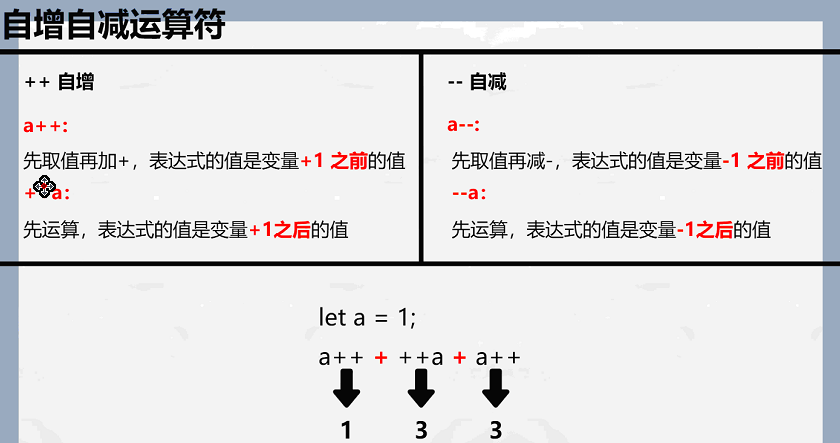
**/= a = a / 5**

**%= a = a % 5**

**把赋值号 = 后面看做一个整体，给变量赋值 a+=b a = a + b**

**自增自减运算符**

++ --



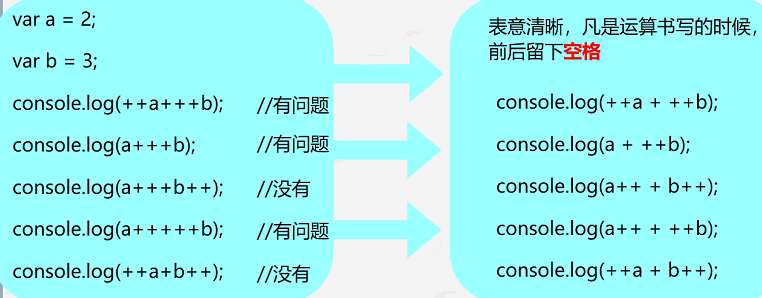
**a ++ 前自增 a-- 前自减**

**++a 后自增 --a 后自减**

**a++（a--）先取值再加（减），表达式的值是变量前加（减）得值**

**++a (--a)先运算，表达式的值是运算后的值**

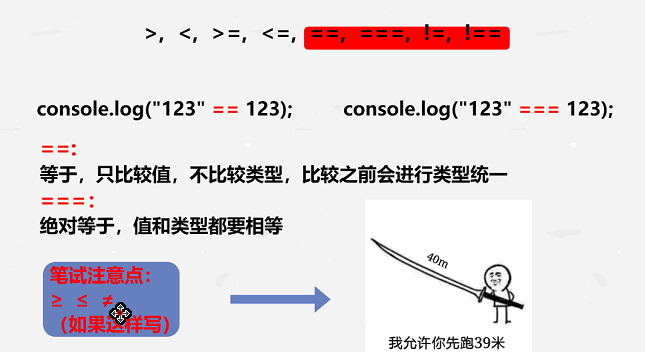
贪心法则：确保用空格把需要计算的表达式隔开，确保表意清晰



**关系运算符**

> < >= <= == === != !==

99%的情况下都用于数值之间的比较，关系运算表达式，始终会解析为Boolean类型（true false）



=== 绝对等于 表示值和类型都要相等

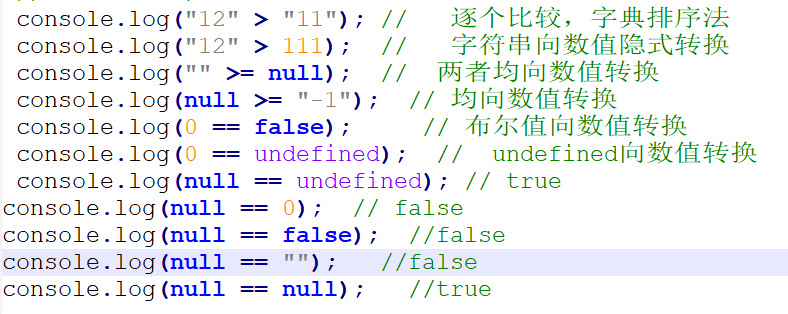
== 等于 表示值相等类型可以不等

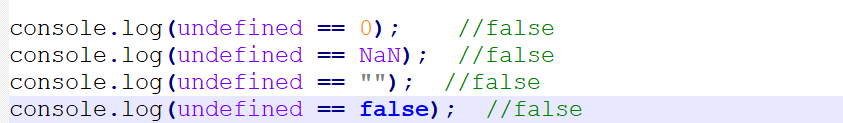
!= 不等于 表示结果与 ==相反

!== 不等于 表示结果与 ===相反

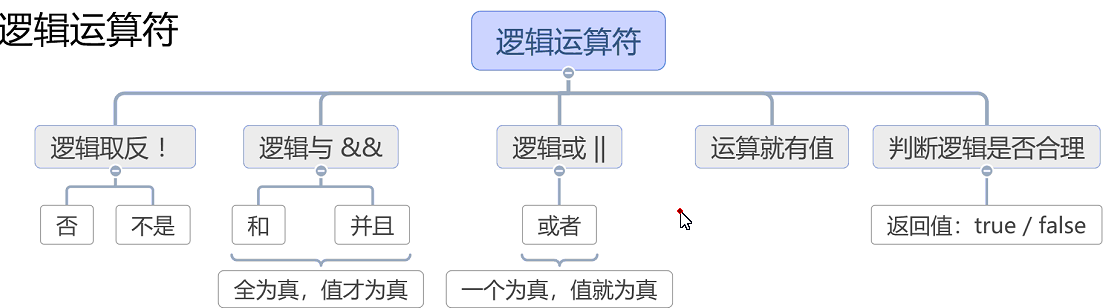
不同类型的数据运算时会发生一系列的隐式类型转换

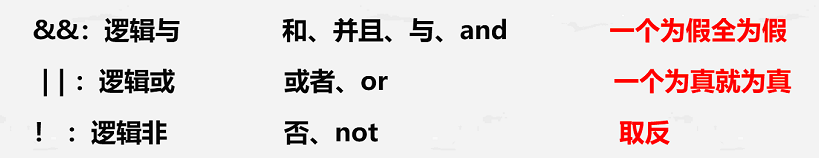
只有null和undefined作比较时是true与其他四个比较均为false





**逻辑运算符**





逻辑与 && 并且、和、and

逻辑或|| 或者、要么

逻辑非！ 否、不行、取反

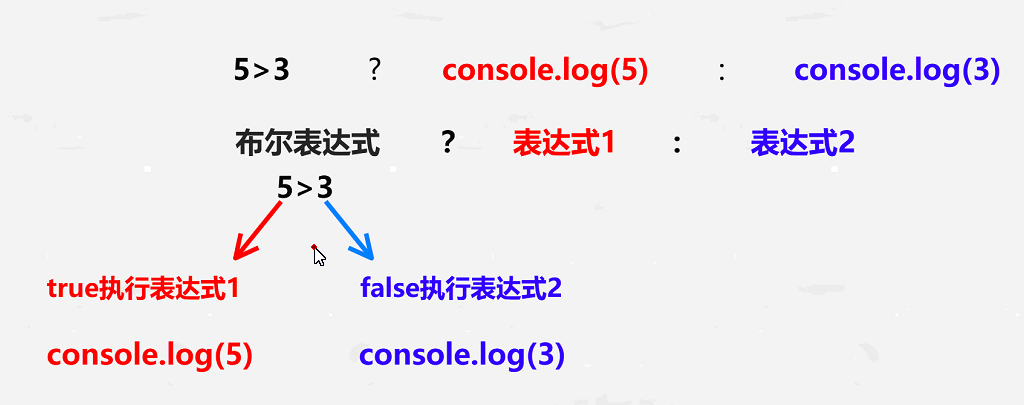
闰年表达公式：

let year = 2012;

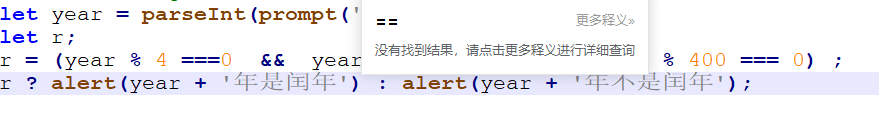
console.log(Boolean((year % 4 ===0 && year %100 !==0) || (year % 400 === 0)));

**三目运算符**

布尔表达式 ？ 表达式1 ： 表达式2

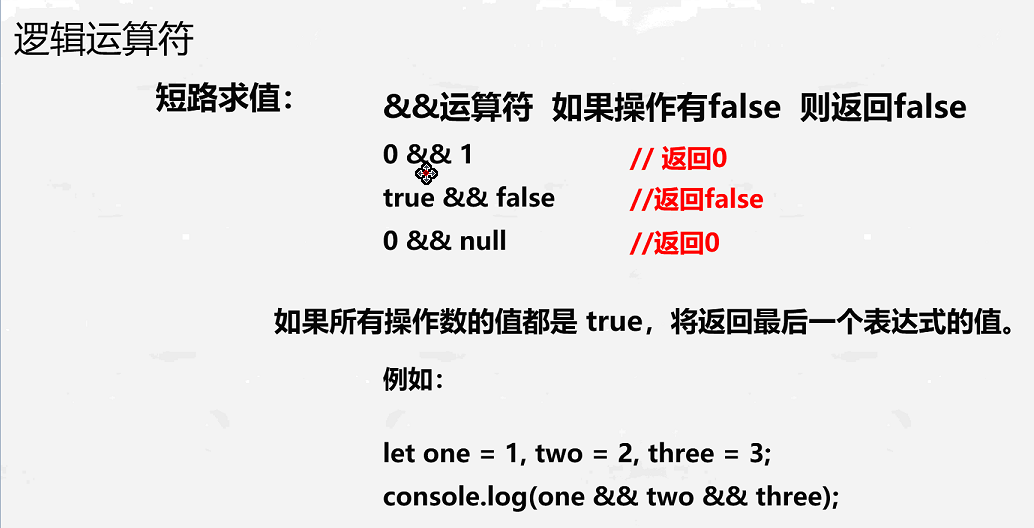


算是否时闰年：



三目运算符其他用法，算闰年里确认是否是世纪年：

(year % 4 ===0 && year %100 !==0) ? alert(year + '年是闰年但不是世纪年') : (year % 4 ===0 && year %100 !==0) || (year % 400 === 0) ? alert(year + '年是闰年,并且是世纪年') : alert(year + '年不是闰年');



**短路求值：**

**&& 有false返回false 全为true则返回最后一个表达式的值。**

**|| 有 true 返回true 全为 false 则返回最后一个表达式的值。**



**位运算符**

进制是一种计数时的一种规则：

N进制 逢N进一，借一当N

二进制 前导符号 0b BIN

八进制 前导符号 0o OCT

十进制 前导符号 无 DEC

十六进制 前导符号 0x HEX

位运算符操作的是底层的二进制数，效率极高

实际开发当中，不建议使用

**1、按位与 & 把二进制数一一对应起来 都为真才为真（都为1才为1） 否则为假**

如果两个相应的二进制位都为1，则该位的结果值为1，否则为0

**2、按位或 | 其一为真就为真 （多用于取整 | 0）数据截断**

如果两个相应的二进制位有一个为1，则该结果为1，否则为0

**3、按位异或 ^ 不一样的才为真 （多用于数据交换）**

如果两个相应的二进制位值不同则为1，否则为0

**4、按位取反 ~ 结果与本身相加得-1**

Js 64位浮点数储存 在运算时不失时机的转为32位的0 1的二进制数

第一位符号位0代表正数，1代表负数

数值在内存当中的时候，存的都是补码，读取的时候又转为源码

* **源码 反码 补码**

正数的源码 反码 补码都是它本身

负数的源码：当成正数转为二进制 把符号位换成1

10000000 00000000 00000000 00000100 源码

11111111 11111111 11111111 11111011 反码：在源码的基础上取反，符号位不变

11111111 11111111 11111111 11111100 补码：在反码的基础上加1

**5、左移 << 移一位放大2倍** **通过从右边推入0，使最左边的位脱落 放大**

**6、右移 >> 移一位缩小0.5倍 通过从左边推入0，使最右边的位脱落 缩小**

**7、无符号右移 >>>通过从右边推入0，使最左边的位脱落 符号位不变**

**交换数据的方法 ：**

// **1.借助第三个变量**

let a = 7; //一瓶可乐

let b = 8; //一瓶雪碧

let c; //找来一个杯子

c = a; //把可乐放进杯子

a = b; //把雪碧放进可乐瓶

b = c; //现在把杯子里的可乐放进雪碧瓶

console.log(a,b);

**第二种方法**：

let a = 7;

let b = 8;

let sum;

sum = a + b;

a = sum - a;

b = sum - a;

console.log(a,b);

**第三种方法：**

let a = 8; // 1000

let b =7 ; //0111

a = a ^ b;

//a（8）与b（7）的异或结果为：1111，转换为10进制为15，a赋值15

b = a ^ b;

//现在a是15 与b（7）异或,1111异或0111，转换为1000，十进制为8，赋值给b

a = a ^ b;

//现在a（15）与b（8）异或，转换为0111，十进制为7，因此a再次被赋值为7，到这里a与b就已经替换过来了。

console.log(a,b);

**第四种方法： 结构赋值**

let a = 8;

let b =7 ;

[a,b] = [b,a];

console.log(a,b);

**数组**

强类型语言定义数组：同一种数据的集合，连续的储存区域

**JS弱类型语言定义数组：数据的集合，连续的存储区域。可以容纳任何数据类型**

Eg :JS [1,2.3sfsf,’d’,{},null,[]]

js里数组的声明方式：

1. **构造函数法：创建一个空数组 let arr = new Array();**
2. **创建一个确定容量的数组 let arr = new Array(10);**
3. **创建一个有数据的数组 let arr = new Array(1,2，‘’大声道，null,false);**
4. **字面量法：创建一个空数组 let arr = [];**
5. **创建一个有数据的数组：let arr = [1,00,'ggg',true];**
6. **创建一个确定容量的数组 let arr.length =10 (通过修改length属性来修改容量)**

Length 属性：

在数组里表示的是数组的容量

字符串也有length属性，但是不能通过动态修改length的值来改变字符串的值 **坑：修改并不会报错，但也不生效**

数组的元素：

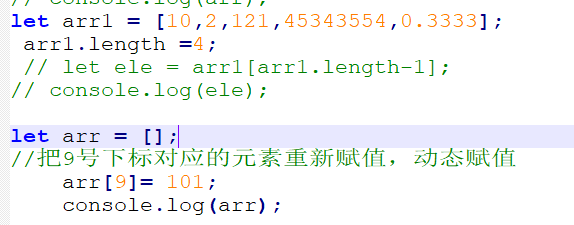
在数组里面，每一个逗号前面的内容的数据称之为元素，

数组的下标：

在数组里，每一个元素在数组中都有一个下标一一对应

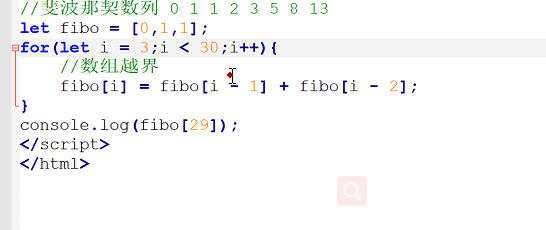
注意：下标从0开始

***如果一个空数组，不想直接给定length，可以通过改变length属性，强制改变数组的容量***



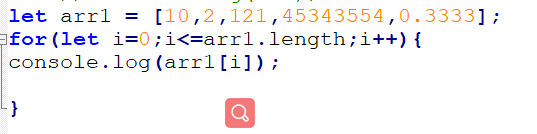
数组越界：返回undefined

往前越界或者往后越界（下标超出数组范围）

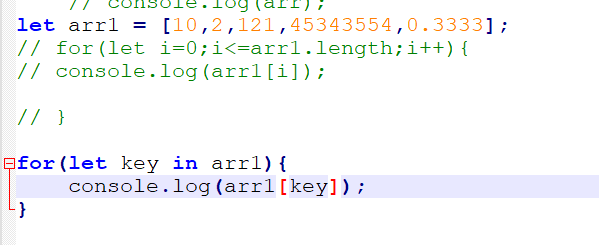


二、数组的遍历

1‘for循环遍历：

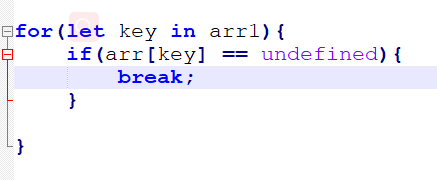


2.for…in…快速遍历

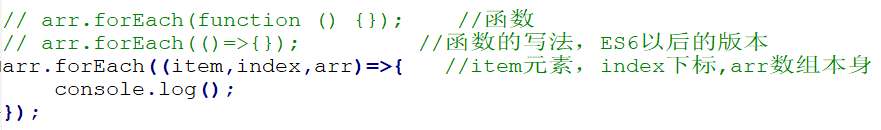


注意：console.log(key) 注：返回的是数组的下标，并且属于字符串形式

console.log(arr1[ket])才是返回元素



3．forEach()遍历



**缺点，****不可以停，一直要遍历完直到遍历结束**

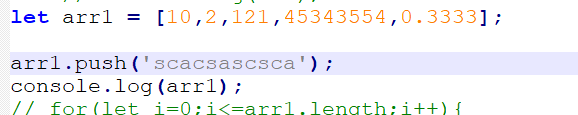
**总结：for循环 for…in…可以通过braek停下**

**forEach()遍历不可以停，一直要遍历完直到遍历结束**

**三、增、删、改、查**

**增加：**

**arr.push(); 往数组的末尾向数组里面推入一个或多个元素，原始数组的长度发生改变。**



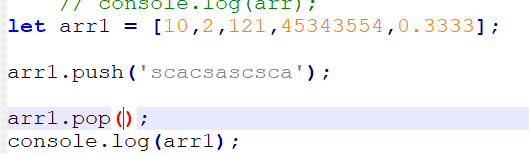


**动态赋值：arr[6]=1;**

**本身没有6下标，给这个6下标对应的位置赋值**

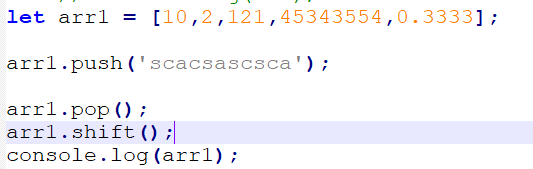
**删：**

**Pop();sort并返回数组的最后一个元素，从数组的末尾删除最后一个**



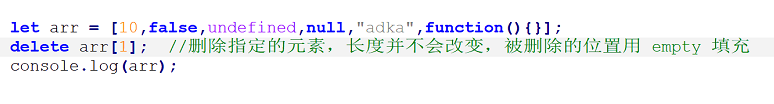


**Shift();删除并返回数组的第一个元素，从数组开头删除一个**





**Delete arr[1];删除指定的元素，长度并不会改变，被删除的位置用empty填充**



**Splice()**



**Splice(a,b,c):**

**A参数表示的是下标，表示需要从哪个下标对应的元素开始删除**

**B表示的是元素的个数，表示需要删除几个元素**

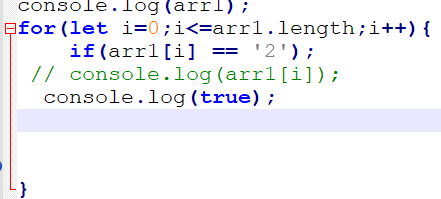
**C表示已经删除的位置需要重新添加的元素。**

**改：**

**拿到需要修改的元素，重新赋值；**

**查：**

**遍历数组作对比的方法：**



**IndexOf();方法**



**返回的是下标，括号里的参数是需要检测的元素**

**返回-1的情况表示数组里没有需要检测的元素。**

**数组去重：**

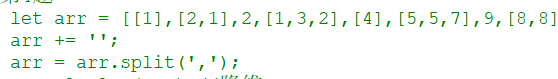


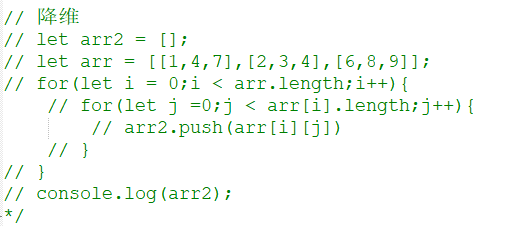


**数组降维：**

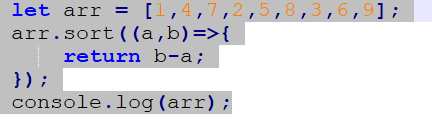
**Arr\_flat = arr.flat(infinity);**

**或者：**





**数组里元素的排序（从小到大）**



**字符串转换为数组：**

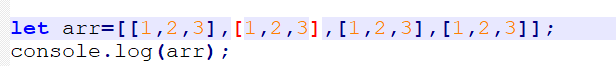


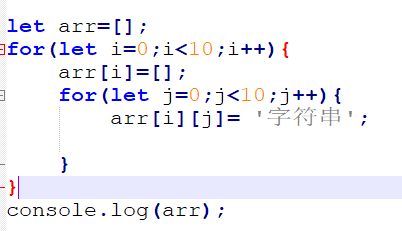
**数组转换为字符串：**

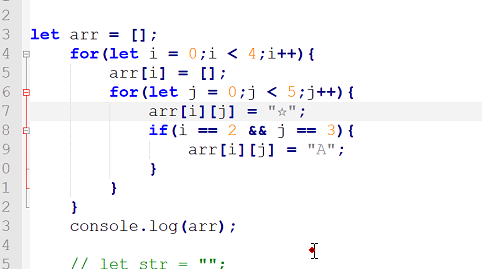


**二维数组**

Let arr = [[ ],[ ],[ ],[ ]];







输出矩阵，选择某个位置更改元素

**函数**

**函数是由事件驱动的，或者当它被调用时的时候执行，可重复执行使用的代码块。**

**函数的声明方式：**

**关键字 函数名 （参数）{函数体/代码块}**

**Function add () { }**

**第二种声明方式：(ES5.1)**

**Let add = function () { }**

**第三种：(ES6)**

**Let add = () => { }**

**第四种：**

**Let add = Function(**

**‘x’,**

**’y’,**

**‘return x + y ’**

**)**

**等同于：**

1. **function add (x,y){**

**return x + y;**

**}**

**函数才是“一等公民”：**

**体现在：1函数是可以重复使用的代码块**

**2函数可以在声明之前调用**

**3js可以把函数看做一种值，与其他的（数值、字符串、布尔值）一样，凡是用值得地方，都可以用函数代替（体现出多功能性）；**

**Return返回值，并且结束。**

**函数的用参数来做区分可以分为四类**

**有参有返 有参数，也有返回值。**

**Function add (x,y){**

**Return x+y;**

**}**

**有参无返 有参数，没有返回值**

**Let toStr = function(a){**

**A+=’’;**

**}**

**无参有返 无参数，有返回值**

**Math.random()**

**无参无返 无参数，没有返回值**

**Let arr=[1,2,3,4,5];**

**Let fun = function(){**

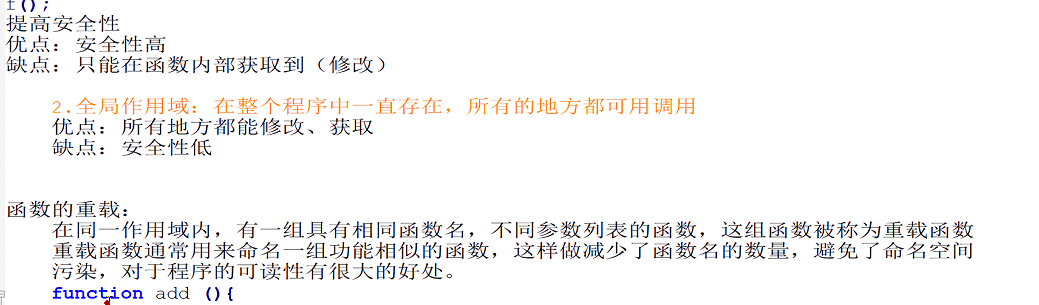
**Arr[3]=100**

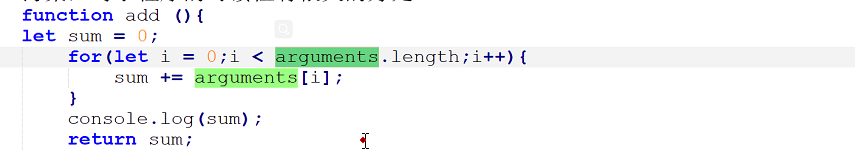
**}**

**Fun();**

**函数的重载**

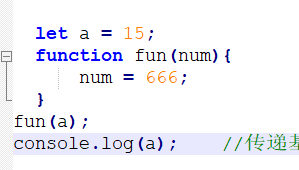
**同一个函数完成不同的功能。这就是重载函数。重载函数常用来实现功能类似而所处理的数据类型不同的问题。不能只有函数返回值类型不同**。



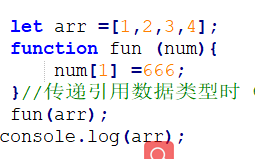


**函数传值的方式：**

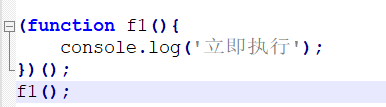
**1.传递基本数据类型时，在函数内部修改，不会对这个值造成影响。 传值**



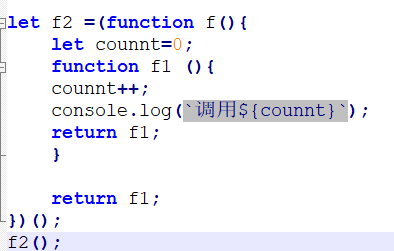
**2.** **传递引用数据类型时（数组，对象）时，在函数内部修改，会对这个值造成影响和改变 传址**

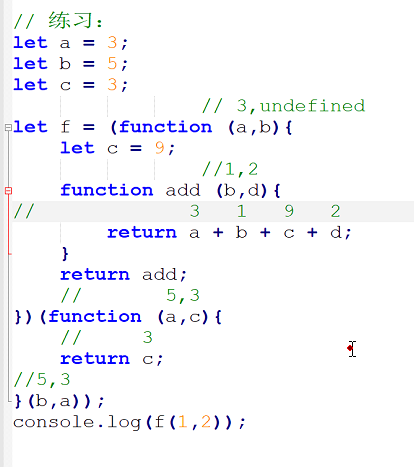


**立即执行函数:创建完成之后立即执行并立即执行一次，并销毁的函数**

**再次调用报错**

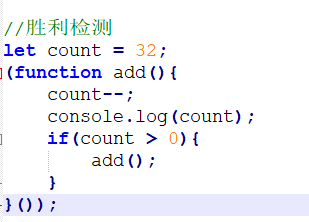
**闭包函数**





**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*递归函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**递归：自己调用自己**



**键值对**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**狭义的对象：键值对**

**广义的对象：面向对象 万物皆对象**

**键值对：**

**定义：储存的是数据库最简单的组织形式**

**键：存储值得编号**

**值：存储的数据**

**对：成对出现的，一个键对应一个值，一一对应**

**对象名**

**Let hero = {**

**键名 属性**

**Name ： “黄鸡”，**

**Level ：3，**

**Ap : 30,**

**Mp :0**

**}**

**键值对相较于数组（数组是以下标的形式确定里面的元素，键值对是以键名来确定属性的值）**

**创建方式：**

**Let obj = {**

**注意点：1每一条属性中间，以逗号，隔开**

**2键名的命名规则 1）键名都会默认的转换为字符串**

**2）键名如果符合标识符命名规则的可以不加“”**

**3）键名允许出现数字键（纯数字的键名）**

**4）以数字开头拼接其它字符的不满足条件**

**5）可以使用关键字**

**6）键名必须有匹配的键值，保证一一对应。**

**键名1 ：键值1,**

**键名2 ：键值2,**

**键名3 ：键值3,**

**键名4 ：键值4,**

**}**

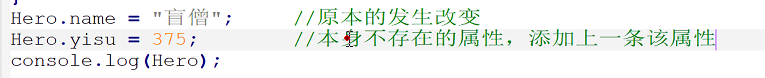
**键值对的访问方式：**

1. **对象名[属性名] 只能用于访问数字键**
2. **对象名[‘属性名’] 百分之百成功，一定能访问**
3. **对象名.属性名 只能用于符合标识符命名规则的属性名（不会产生歧义，数字键不会成功）**

**属性的增删改查**

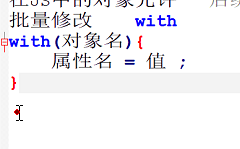
**增、改**

**如果是本身存在的属性，改变赋值时会发生改变，如果是不存在的属性，则在该对象中添加一条**



**在JS中的对象允许“后绑定” 也叫动态赋值**

**批量修改：(需要注意的是，分号结尾并且修改的属性名必须是原本就存在的属性)**

**With（对象名）{**

**Name = “提莫”;**

**Level = 6;**

**}**

**在做属性值得修改时，属性名必须遵循标识符命名规则，在对象里不存在的属性，不能通过with、直接添加（不报错，也添加不上）**

**删除**

**Delete Hero[‘name’]; 删除某一属性删除已有的属性，正常删除，删除没有的属性 不会报错**

**写一堆在里面的时候，只能删除最后一项属性**

**查**

**获取一个对象所有的键名**

 **Object.keys(对象名)；**

**获取到所有的键名，并以数组的形式呈现**

**Object.getOwnPropetyNames(对象名)；**

**获取到所有的键名，并以数组的形式呈现**

**检查属性是否存在：**

**In ( 运算符) ；检查的是键名，检测对象里是否包含某一条属性 包含就返回true 不包含就为flase**

**写法:**

**属性名 in 对象名**

**console.log('666' in Huangji);**

**hasOwnProperty用来检测哪些属性是自身的属性**

**写法 ：对象名. hasOwnProperty(属性名);**

**Console.log(Huangji. hasOwnProperty(‘666’));**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*对象\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**引用数据类型 object**

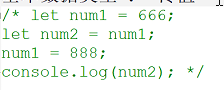
**狭义的对象：键值对{}**

**广义的对象：指的是一系列的属性和功能或者说是方法的集合（在js里，object是所有数据类型的老祖宗）**

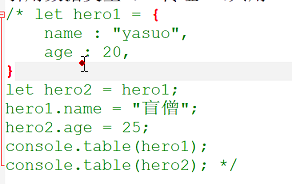
**基本数据类型与引用数类型之间的区别：**

**传值的方式：**

**基本数据类型 ： 传值**



**引用数据类型： 传址（共用一个仓库） 地址**



**Let obj = JSON.stringify(hero1);**

**将对象转换为字符串**

**加密操作JSON.stringify()**



**Let hero1 = JSON.parse(obj)**

**将字符串转换为对象**

**解密操作JSON.parse()**



**JSON**

**方法：**

1. **静态方法：在对象本身上的方法：**

**Math JSON**

1. **动态的方法**

**.toString .push .sort**

**\*\*\*\*\*\*\*\*\*\*\*\*面向对象\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**面向对象三种特征**

***封装*：**

**隐藏对象的属性和实现的细节，仅对外公开接口，控制在程序中属性的读写和修改的一些访问级别 ‘类’**

***继承*：**

**字面意思，继承。**

***多态*：**

**把相同的消息发送给不同的对象时，会引发不同的动作**

**面向对象 万物皆对象 抽象性的概念**

**需求 实现**

**模板 成品**

**抽象 具象**

**需求：还不存在的东西**

**实现：实实在在存在的东西**

**实例化对象的方式之一：(原始模式)**

**Let hero = {**

**Name : ‘葫芦娃’;**

**Age : 5;**

**}**

**原始模式改进版**

**Function hero（name.age）{**

**Return {**

**Name:name,**

**Age:age,**

**}**

**}**

**Let ys = hero(‘亚索’,18);**

**Let xz =hero(‘李青’,20);**

**Console.log(ys,xz);**

**实例化对象方法之一工厂模式**

**Function hero(name,age){**

**Let obj = { }; //原材料阶段**

**Obj.name = name; //加工阶段**

**Obj,age = age;**

**Return obj； //把成品输出**

**}**

**实例化对象的方式之一: 构造函数法**

**Let People = function(name,age,sex,KG){**

**This.name =name;**

**This.age=age;**

**This.sex =sex;**

**This.KG =KG;**

**}**

**Let ZhangSan = new People(‘张三’，‘23’，‘未知’，‘66’);**

***This* 指向的是正在被构建的对象**

**写构造函数时，需要注意的点**

1. **构造函数的函数名首字母大写，普通函数首字母小写**
2. **构造函数内部使用的this关键字，this指向的是正在被构建的对象**
3. **构造函数调用时，需要使用new关键字**

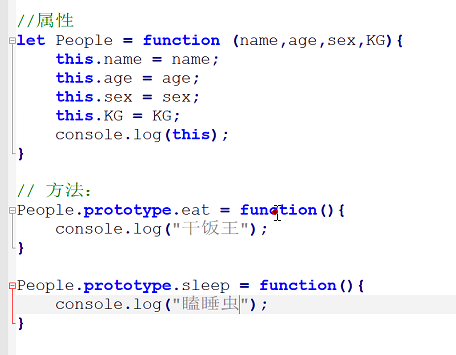
**New关键字**

1. **创建一个空对象**
2. **将空对象赋值给构造函数内部的this**
3. **执行构造函数内部的代码**
4. **将空对象的原型指向构造函数的原型**

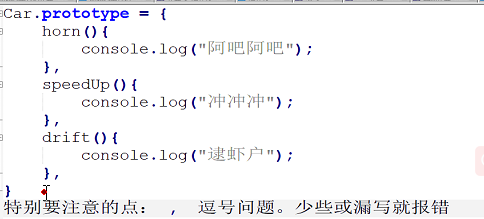
**构造函数怎么写？**

1. **属性写在自身上**
2. **方法写在原型上，因为实例的对象公用一个方法，这么做节省空间**

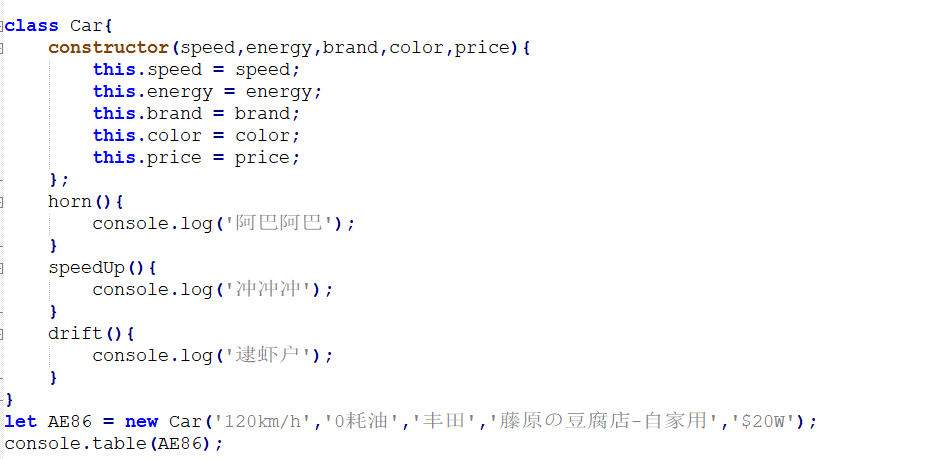
**ES5.1的写法**





**方法二**

**ES6的写法：**



**异步函数**

**进程：（进行的过程，某件事务从发生到结束的过程）**

**在计算机中**

**进程:正在执行的程序就是一个程序**

**线程：进程中一个单一的连续的控制流程。**

**一个进程可以同时有多个线程**

**单线程：进行一个程序时，同时执行一条命令**

**优点：顺序编码简单**

**缺点：效率低下**

**多线程：进行一个程序时，同时执行多条命令**

**优点：提高CPU利用率**

**缺点：线程的并发问题、线程切换以及线程创建的时间开销**

**坐公交先上先刷卡，后上后刷卡称为同步任务**

**但是到你的时候你卡暂时刷不上，司机叫你先站一边，过一会你又刷上了，你刷卡与其他人不同步，就称之为异步任务**

**在游戏开发中实际运用到异步函数——计时器**

1. **隔多少毫秒之后执行一次，只要有这个异步函数就会被挪到辅线程等待，等正常的程序执行完毕再执行异步函数**

**写法：**

**SetTimeout(()=>{**

**//需要执行的代码**

**Console.log(111);**

**},5000);**

**第二种：**

**setTimeout(function(){**

**});**

1. **每隔多少毫秒执行一次**

**setInterval ((111)=>{**

**//需要执行的代码**

**Console.log(111);**

**});**

**setInterval(function(){**

**});**

**清除计时器**

**Let count = 0;**

**Let a = setInterval(()=>{**

**Console.log(666);**

**Count++;**

**If(count ==5){**

**clearInterval(a);**

**//clearTimeout(a); //用的哪个写哪个**

**}**

**},1000)**