快速入门：https://www.liaoxuefeng.com/wiki/1022910821149312/1023020895584256

在浏览器上输入

var height = parseFloat(prompt('请输入身高(m):'));

var weight = parseFloat(prompt('请输入体重(kg):'));

重点笔记：.JS严格区分大小写

//数据类型

Number

NaN //Not a Number ，无法计算结果时，用NaN表示

JavaScript不区分整数和浮点数，统一用Number表示，以下都是合法的Number类型：

123; // 整数123

0.456; // 浮点数0.456

1.2345e3; // 科学计数法表示1.2345x1000，等同于1234.5

-99; // 负数

NaN; // NaN表示Not a Number，当无法计算结果时用NaN表示

Infinity; // Infinity表示无限大，当数值超过了JavaScript的Number所能表示的最大值时，就表示为Infinity

判断是否相等：a===b

NaN === NaN; // false

唯一能判断NaN的方法是通过isNaN()函数：

isNaN(NaN); // true

var a = 123; //a就是整型

a = 'ABC'; //a就是字符串，这叫动态类型

int a = 123; //a是整型

a = 'ABC'; //报错，字符串不能赋给整型，这叫静态类型

strict模式

JavaScript在设计之初，为了方便初学者学习，并不强制要求用var申明变量。这个设计错误带来了严重的后果：如果一个变量没有通过var申明就被使用，那么该变量就自动被申明为全局变量：

i = 10; // i现在是全局变量

在同一个页面的不同的JavaScript文件中，如果都不用var申明，恰好都使用了变量i，将造成变量i互相影响，产生难以调试的错误结果。

使用var申明的变量则不是全局变量，它的范围被限制在该变量被申明的函数体内（函数的概念将稍后讲解），同名变量在不同的函数体内互不冲突。

为了修补JavaScript这一严重设计缺陷，ECMA在后续规范中推出了strict模式，在strict模式下运行的JavaScript代码，强制通过var申明变量，未使用var申明变量就使用的，将导致运行错误。

启用strict模式的方法是在JavaScript代码的第一行写上：

'use strict';

这是一个字符串，不支持strict模式的浏览器会把它当做一个字符串语句执行，支持strict模式的浏览器将开启strict模式运行JavaScript。

反引号可以一次性打多行；

console.log(`多行

字符串

测试`);

var name = 20;

${ name } === 20;

$可以直接取出变量的内容；

对象

JavaScript的对象是一组由键-值组成的无序集合，例如：

**var** person = {

name: 'Bob',

age: 20,

tags: ['js', 'web', 'mobile'],

city: 'Beijing',

hasCar: true,

zipcode: null

};

JavaScript对象的键都是字符串类型，值可以是任意数据类型。上述person对象一共定义了6个键值对，其中每个键又称为对象的属性，例如，person的name属性为'Bob'，zipcode属性为null。

要获取一个对象的属性，我们用对象变量.属性名的方式：

person.name; // 'Bob'

person.zipcode; // null

如果我们要检测xiaoming是否拥有某一属性，可以用in操作符：

var xiaoming = {

name: '小明',

birth: 1990,

school: 'No.1 Middle School',

height: 1.70,

weight: 65,

score: null

};

'name' **in** xiaoming; // **true**

'grade' **in** xiaoming; // **false**

不过要小心，如果in判断一个属性存在，这个属性不一定是xiaoming的，它可能是xiaoming继承得到的：

'toString' **in** xiaoming; // **true**

因为toString定义在object对象中，而所有对象最终都会在原型链上指向object，所以xiaoming也拥有toString属性。

要判断一个属性是否是xiaoming自身拥有的，而不是继承得到的，可以用hasOwnProperty()方法：

var xiaoming = {

name: '小明'

};

xiaoming.hasOwnProperty('name'); // **true**

xiaoming.hasOwnProperty('toString'); // **false**

操作字符串

调用这些方法不会改变原来的字符串，会返回一个新的字符串

**var** s = 'Hello';

s.toUpperCase(); *// 返回'HELLO'*

**var** s = 'Hello';

**var** lower = s.toLowerCase(); *// 返回'hello'并赋值给变量lower*

lower; *// 'hello'*

indexOf()会搜索指定字符串出现的位置：

**var** s = 'hello, world';

s.indexOf('world'); *// 返回7*

s.indexOf('World'); *// 没有找到指定的子串，返回-1*

substring()返回指定索引区间的子串：

**var** s = 'hello, world'

s.substring(0, 5); *// 从索引0开始到5（不包括5），返回'hello'*

s.substring(7); *// 从索引7开始到结束，返回'world'*

var x. charCodeAt(); //字符x的ASCII码

数组

数组可以包含任意类型，通过下标来访问某个元素

要取得Array的长度，直接访问length属性：

**var** arr = [1, 2, 3.14, 'Hello', null, true];

arr.length; *// 6*

请注意，直接给Array的length赋一个新的值会导致Array大小的变化：

**var** arr = [1, 2, 3];

arr.length; *// 3*

arr.length = 6;

arr; *// arr变为[1, 2, 3, undefined, undefined, undefined]*

arr.length = 2;

arr; *// arr变为[1, 2]*

Array可以通过索引把对应的元素修改为新的值，因此，对Array的索引进行赋值会直接修改这个Array：

**var** arr = ['A', 'B', 'C'];

arr[1] = 99;

arr; *// arr现在变为['A', 99, 'C']*

请注意，如果通过索引赋值时，索引超过了范围，同样会引起Array大小的变化：

**var** arr = [1, 2, 3];

arr[5] = 'x';

arr; *// arr变为[1, 2, 3, undefined, undefined, 'x']*

大多数其他编程语言不允许直接改变数组的大小，越界访问索引会报错。然而，JavaScript的Array却不会有任何错误。在编写代码时，不建议直接修改Array的大小，访问索引时要确保索引不会越界。

### indexOf

与String类似，Array也可以通过indexOf()来搜索一个指定的元素的位置：

**var** arr = [10, 20, '30', 'xyz'];

arr.indexOf(10); *// 元素10的索引为0*

arr.indexOf(20); *// 元素20的索引为1*

arr.indexOf(30); *// 元素30没有找到，返回-1*

arr.indexOf('30'); *// 元素'30'的索引为2*

注意了，数字30和字符串'30'是不同的元素。

slice

slice()就是对应String的substring()版本，它截取Array的部分元素，然后返回一个新的Array：

**var** arr = ['A', 'B', 'C', 'D', 'E', 'F', 'G'];

arr.slice(0, 3); *// 从索引0开始，到索引3结束，但不包括索引3: ['A', 'B', 'C']*

arr.slice(3); *// 从索引3开始到结束: ['D', 'E', 'F', 'G']*

注意到slice()的起止参数包括开始索引，不包括结束索引。

如果不给slice()传递任何参数，它就会从头到尾截取所有元素。利用这一点，我们可以很容易地复制一个Array：

**var** arr = ['A', 'B', 'C', 'D', 'E', 'F', 'G'];

**var** aCopy = arr.slice();

aCopy; *// ['A', 'B', 'C', 'D', 'E', 'F', 'G']*

aCopy === arr; *// false*

### push和pop

push()向Array的末尾添加若干元素，pop()则把Array的最后一个元素删除掉：

**var** arr = [1, 2];

arr.push('A', 'B'); *// 返回Array新的长度: 4*

arr; *// [1, 2, 'A', 'B']*

arr.pop(); *// pop()返回'B'*

arr; *// [1, 2, 'A']*

arr.pop(); arr.pop(); arr.pop(); *// 连续pop 3次*

arr; *// []*

arr.pop(); *// 空数组继续pop不会报错，而是返回undefined*

arr; *// []*

### unshift和shift

如果要往Array的头部添加若干元素，使用unshift()方法，shift()方法则把Array的第一个元素删掉：

**var** arr = [1, 2];

arr.unshift('A', 'B'); *// 返回Array新的长度: 4*

arr; *// ['A', 'B', 1, 2]*

arr.shift(); *// 'A'*

arr; *// ['B', 1, 2]*

arr.shift(); arr.shift(); arr.shift(); *// 连续shift 3次*

arr; *// []*

arr.shift(); *// 空数组继续shift不会报错，而是返回undefined*

arr; *// []*

### sort

sort()可以对当前Array进行排序，它会直接修改当前Array的元素位置，直接调用时，按照默认顺序排序：

**var** arr = ['B', 'C', 'A'];

arr.sort();

arr; *// ['A', 'B', 'C']*

### splice

splice()方法是修改Array的“万能方法”，它可以从指定的索引开始删除若干元素，然后再从该位置添加若干元素：

**var** arr = ['Microsoft', 'Apple', 'Yahoo', 'AOL', 'Excite', 'Oracle'];

*// 从索引2开始删除3个元素,然后再添加两个元素:*

arr.splice(2, 3, 'Google', 'Facebook'); *// 返回删除的元素 ['Yahoo', 'AOL', 'Excite']*

arr; *// ['Microsoft', 'Apple', 'Google', 'Facebook', 'Oracle']*

*// 只删除,不添加:*

arr.splice(2, 2); *// ['Google', 'Facebook']*

arr; *// ['Microsoft', 'Apple', 'Oracle']*

*// 只添加,不删除:*

arr.splice(2, 0, 'Google', 'Facebook'); *// 返回[],因为没有删除任何元素*

arr; *// ['Microsoft', 'Apple', 'Google', 'Facebook', 'Oracle']*

### concat

concat()方法把当前的Array和另一个Array连接起来，并返回一个新的Array：

**var** arr = ['A', 'B', 'C'];

**var** added = arr.concat([1, 2, 3]);

added; *// ['A', 'B', 'C', 1, 2, 3]*

arr; *// ['A', 'B', 'C']*

请注意，concat()方法并没有修改当前Array，而是返回了一个新的Array。

实际上，concat()方法可以接收任意个元素和Array，并且自动把Array拆开，然后全部添加到新的Array里：

**var** arr = ['A', 'B', 'C'];

arr.concat(1, 2, [3, 4]); *// ['A', 'B', 'C', 1, 2, 3, 4]*

### join

join()方法是一个非常实用的方法，它把当前Array的每个元素都用指定的字符串连接起来，然后返回连接后的字符串：

**var** arr = ['A', 'B', 'C', 1, 2, 3];

arr.join('-'); *// 'A-B-C-1-2-3'*

如果Array的元素不是字符串，将自动转换为字符串后再连接。