## JS引用

1. Script标签一旦用于引入外部文件，就不能再编写代码了。需再创建一个新的script标签用于编写。
2. 可放在a标签和事件中



## 变量

1. 在js中，需要创建一个变量，需要var 声明

var a;

1. 为变量赋值

a=123;

或 var b=456;

## 标识符

在js中所有由我们自主命名的都可以称为标识符

例如：变量名，函数名，属性名

命名规则：

1. 标识符中可以含有字母、数字、\_ 、$
2. 标识符不能以数字开头
3. 标识符不能使用关键字和保留字（如：if、var）
4. 标识符一般采用驼峰命名法

var a\_1\_$;

## string

1. 使用字符串时引号不能嵌套：一对单引号中不能出现单引号，双引号同理。若需使用，则要用\转义字符。如： str=”我说:\”今天天气真好\””;

\n表示换行

\t 制表符（tab）

\\ 表示 \

1. 可以使用typeof来检查一个变量的类型

语法：typeof 变量

1. JS中可以表示的数字最大值：Number.MAX\_VALUE。如果使用number表示的数字超过了最大值，则会返回Infinity,表示正无穷，Infinity也属number。
2. 若使用js进行浮点数运算，可能得到一个不精确的结果。如：c = 0.1+0.2
3. 强制类型转换：
4. a = string(a);
5. a = a.toString(); //在转换null和 undefined会报错

## number

1. 使用Number（）函数
2. 如果是纯数字的字符串，则直接转换为数字
3. 如果字符串中有非数字的内容，则转换为NaN
4. 如果字符串是一个空串或者是一个全是空格的字符串，则转换为0
5. 使用parseInt()、parsefloat()

 

## 其它进制的数字

1. 在js中，如果需要表示16进制的数字，则需要以0x开头(并不是所有浏览器都支持)

 

1. 如果需要表示8进制的数字，则需要以0开头



1. 如果需要表示2进制的数字，则需要以0b开头



## 转换为boolean

1. 数字：除了0和NaN，都是true



1. 字符串：除了空串，都是true
2. null 和undefined都是false

## 运算符

1. 任何值和NaN做运算都得NaN
2. 任何值（包括NaN）和字符串（包括空串）相加都会被转换为字符串



1. 任何值做- \* /运算时都会自动转换为Number
2. 快速转换为number



## 自增和自减

1. a++和++a都会立即使原变量的值加一。但a++的值等于原变量的值（即等于a），++a等于自增后的值（即a+1）
2. 同上，a--的值等于原变量的值（即等于a），--a等于自增后的值（即a-1）

## 逻辑运算符

1. && （与）

找false，从左至右运行，当存在false，后面不再运行



1. || （或）

找true，从左至右运行，当存在true，后面不再运行



1. true && true ，当两个值都为true，则返回后边的

false && false ，当两个值都为false，则返回前面的

（若第一个值为true，则必然返回第二个值；若第一个值为false，则必然返回第一个值）





1. true || true ，当两个值都为true，则直接返回前面的

false || false ，当两个值都为false，则直接返回后面的

（若第一个值为true，则直接返回第一个值；若第一个值为false，则必然返回第一个值）

## 关系运算符

1. 若两个字符串相比较，则只会取第一位进行字符编码比较





## Unicode编码

1. 在字符串中使用转义字符\u 可以输出Unicode编码



1. 在HTML中，使用&#可输出Unicode编码

（Unicode编码默认为16进制，在html中使用需先转换为10进制）



## 相等运算符

1. 当使用==来比较两个值时，如果两个值的类型不同，则会自动进行类型转换，将其转换为相同类型，然后再比较
2. NaN不和任何值相等
3. != 判断两个值是否不相等
4. === 判断两个值是否全等



1. !== 判断两个值是否不全等



## 条件运算符（三元运算符）

1. 语法：

条件表达式?语句1：语句2;

1. 执行流程：

首先对条件表达式进行求值

如果该值为true，则执行语句1，并返回执行结果

如果该值为flase，则执行语句2，并返回执行结果



1. 如果条件表达式为非布尔值，则会先将其转换为布尔值

## prompt（）

该函数可以弹出一个提示框，并带有一个文本框，可输入一段内容

## 条件表达式

1. 语法：

switch(条件表达式){

case 表达式1：

语句1 。。。

break;

case 表达式2：

语句2 。。。

break;

case 表达式3：

语句3 。。。

break;

default:

语句。。。

break;

}



## for循环



## break和continue

1. break可结束离得最近的整个循环
2. continue只会结束当次循环



当 i=3 时跳过

1. 可以为循环语句创建一个label，来标识循环语句，break后面跟label可结束指定循环



## console.time(“str”)

可开启一个计时器，console.timeEnd("str");用来结束一个计时器



## 对象的分类

1. 内建对象

由ES标准中定义的对象：Math String Number Boolean Function Object…

1. 宿主对象

由JS运行环境提供的对象，主要由浏览器提供：Bom Dom

1. 自定义对象

由开发人员自己创建的对象



## 对象的属性名和属性值

1. 如果要使用特殊的属性名，则需要使用特殊方式：对象[“属性名”] = 属性值



1. 变量



1. 属性值可以是任意数据类型，以及可以是一个对象
2. in可以检查对象中是否含有某个属性



## 基本数据类型和引用数据类型

1. 基本数据类型的值直接在栈内存中存储，值与值之间相互独立



1. 引用数据类型（对象）是保存到堆内存中，变量保存的是对象的内存地址



1. b修改了对象名，两者断开联系



1. 不同对象，即使属性值相同，也不全等（因内存地址不同）



## 对象字面量

1. 可使用{}来直接创建对象



1. 可直接指定对象中的属性（如果使用特殊名，必须加引号）



## 函数

1. 函数生成与调用



1. 可以在（）中指定多个形参



1. 实参可以是任意的数据类型，也可以是一个对象

当参数过多时，可以将参数封装到一个对象中，然后通过对象传递



1. 实参也可以是一个函数



## 定义一个函数，可以根据半径计算一个圆的面积，并返回计算结果



## 立即执行函数

1. 无需调用，立即执行，但也无法再次调用



## 枚举对象中的属性

* + 1. 语法：

for (var 变量 in 对象){

}



## 作用域

1. 全局作用域

- 直接编写在script中的JS代码，都在全局作用域

- 全局作用域中有一个全局对象window，它代表的是浏览器的一个窗口

- 在全局作用域中创建的变量都会作为window对象的属性保存

1. 函数作用域

- 调用函数时创建函数作用域，函数执行完毕后，作用域销毁

- 每调用一次就会创建一个新的作用域，他们之间相互独立

- 在函数作用域中，如若像访问与函数作用域中相同名字的全局作用域变量，需调用window.

## 变量的声明提前

* + 1. 使用var声明的变量，会在所有的代码执行之前被声明（但是不会赋值）



## 方法

函数也可以称为对象的属性，如果一个函数作为对象的属性保存，那么称这个函数是这个对象的方法，调用函数就说调用对象的方法

## this

解析器每次在调用函数时都会向函数内部传递一个隐含参数

这个隐含参数就是this，this指向的是一个对象

这个对象外面称为函数执行的上下文对象

根据函数的调用方式的不同，this也会指向不同的对象

 

## 使用工厂方法创建对象

使用该方法可以大批量创建对象



## 构造函数

1. 构造函数就是一个普通的函数，创建方式和普通函数相同，不同的是构造函数习惯上首字母大写
2. 构造函数的执行流程：
3. 立即创建一个新的对象
4. 将创建的对象设置为函数中的this，在构造函数中可以使用this来引用新建的对象
5. 逐行执行函数中的代码
6. 将新建的对象作为返回值返回



1. 构造函数每执行一次，就会创建一个新的sayName方法，占用空间



1. 使用instanceof 可以检查一个实例是否属于一个类



1. this的情况
   1. 当以函数的形式调用时，this是window
   2. 当以方法的形式调用时，谁调用方法this就是谁
   3. 当以构造函数调用时，this就是新建的那个对象

## 原型prototype

1. 我们所创建的每一个函数，解析器都会向函数中添加一个属性prototype,这个属性对应着一个对象，这个对象就是原型对象
2. 当函数以构造函数的形式调用，它所创建的对象中都会有一个隐含的属性，指向该构造函数的原型对象，我们可以通过\_\_proto\_\_来访问该属性
3. **原型对象就像一个公共的区域，所有同一个类的实例都有可以访问到这个原型对象，我们可以将对象中共有的内容，统一设置到原型对象中！！！**



直接添加方法：



原型对象：



1. 当我们访问对象的一个属性或方法时，它会在对象自身中寻找，如果有则直接使用，如果没有则会去原型对象中寻找。
2. 原型对象中的方法共用一个区域