**（1） 标识符**

**1.标识符是代码中用来标识变量、函数、或属性的字符序列**

命名规则：由字母、数字、下划线和$符号组成；不能以数字开头；大小写敏感（区分大小写）

注意：①标识符不能和 JavaScript 其它关键字同名（如void、this、if等）；②保留字在某种意思上是为将来的关键字而保留的单词，因此保留字不能被用作变量名或函数名（如class、import等）

**2.访问属性的方式**

① 通过点号(.)运算符

点号要求后面的属性名是合法的标识符，对于不合法的不可以使用。

例如person.name

② 通过中括号([])运算符

中括号要求的则是一个字符串即可，不必是合法的标识符

例如person[“first name”]

对象foo访问att属性：foo[“att”]、foo[“a”+”t”+”t”]、foo.att

[]内必须为字符串

当以纯数字命名时，可省””，例如person[0]会发生隐式类型转换成person[“0”]

**3.Window对象的属性**

全局变量是window对象的属性，如window.history、window.location等

全局函数是window对象的方法，如window.alert()、window.clearInterval()等

在定义全局变量时不能与window属性冲突

**（2） 表达式与运算符**

**1. 运算符**

运算符的优先级决定了表达式中运算执行的先后顺序，优先级高的运算符最先被执行

**2. 字面量（直接量）**

字面量，就是表示自身的常量

例如：12、"hello world"、{x:1,y:2}、[1,2,5,6]

**3. 表达式**

运算符+操作数，表达式将产生一个值，用于需要值的地方

**4. 函数表达式**

function add(a,b){

return a+b;

}//函数声明

var add = function add(a,b){

return a+b;

};//函数表达式

JavaScript 解析器识别函数声明的条件是以 function 关键字开始，**只要在** **function** **关键字的前面有任何其他的元素**，就会从函数声明转变为函数表达式

!function(){} —— false

+function(){} —— NaN

(function(){}) —— function(){}

函数调用表达式返回值:

无return语句——返回undefined

return语句后不带表达式——返回undefined

Return语句后带表达式——返回表达式的值

在 function 前面加！、+、 - 甚至是逗号等到都可以起到识别为函数表达式的效果

在这些运算符中加括号是最安全的做法，因为它不会改变函数的返回值。

**5. 逻辑运算符**

① 逻辑运算符两边的操作数都是布尔类型

对于&&来说， 除了两侧都为真时为真，其他情况都为假

对于||来说， 除了两侧都为假时为假，其他情况都为真

② 当逻辑运算符 && 和 || 两侧的操作数不是布尔类型时

首先将左操作数转换成布尔类型

对转换后的左操作数进行逻辑判断（true or false）

根据短路原则返回原始左操作数或原始右操作数

短路原则（忽略对右操作数的判断）：**对于&&，转换后的左操作数若为 true，则直接返回原始右操作数，若为 false 则直接返回原始左操作数；对于||，转换后的左操作数若为 true，则直接返回原始左操作数，若为 false 则直接返回原始右操作数**

③ 短路原则的应用

遵循短路特性，**使用** **||** **来设置函数参数的默认值**

（函数定义时可以给参数指定默认值，调用时若未传参数则该参数的值取它定义时的默认值）

function circlr(r,pi){

return r\*r\*(pi||3.14)

}

遵循短路特性，**使用** **&&** **防止运行报错**

遵循短路特性，**使用** **&&** **和** **||** **可用来实现条件语句**

/\*\*\*使用 &&和|| 可用来实现条件语句\*\*\*/

var score = 76;

if (score > 90) {

console.log("优");

} else if (score > 75) {

console.log("良");

} else if (score > 60) {

console.log("及格");

} else {

console.log("不及格");

}

//通过&&和||的组合实现如上功能，注：小括号优先级最高

console.log(

(score > 90 && "优") ||

(score > 75 && "良") ||

(score > 60 && "及格") ||

"不及格"

);

**6. 相等运算符**

严格相等运算符（===）仅当两个操作数的类型相同且值相等为 true

宽松相等运算符（==）在进行比较之前，将两个操作数转换成相同的类型

**7. 递增递减运算符**

① 递增 (++)

递增运算符为其操作数增加1，返回一个数值

如果使用后置（postfix），即运算符位于操作数的后面（如 x++），那么将会在**递增前返回数值**

如果使用前置（prefix），即运算符位于操作数的前面（如 ++x），那么将会在**递增后返回数值**

② 递减（--）

递减运算符为其操作数减去1，返回一个数值，前置后置与递增相同

var x = 1;

x = x++;//x 等于 x++的返回值1

console.log(x);//1

**8. 赋值运算符**

基于右值（right operand）的值，给左值（left operand）赋值

左值：“=”运算符的左操作数；右值：“=”运算符的右操作数

赋值表达式的返回值为右操作数

function fun() {

var a = b = 5;

console.log(a, typeof a);//5 "number"

console.log(b, typeof b);//5 "number"

}

fun();

console.log(a, typeof a);//undefined "undefined"

console.log(b, typeof b);//5 "number"

var a = {

n: 1

};

var b = a;

a.x = a = {

n: 2

};

console.log(a.x); //undefined

console.log(b); //{n: 1, x: {…}}

var a = {

n: 1

};

a.x = a = {

n: 2

};

console.log(a.x); //undefined

**9. 复合运算符**

x += y —— x = x + y

+ - \* / % 均如此

var x = 2;

x += x++;

console.log(x); //4

x \*= --x;

console.log(x); //12

**10. 逗号操作符**

对它的每个操作数求值（从左到右），并返回最后一个操作数的值