JS语法

语句块

JS使用大括号构成语句块。
ES6 之前语句块是没有作用域的，从ES6开始支持`块作用域`，let只能在块作用域内可见

```
function hello() {
    let a = 1;
    var b = 2;
    c = 3
}

//let d = 100
if (1)
{
    let d = 4; // 去掉let会怎么样？
    var e = 5;
    f = 6
    if (true) {
        console.log(d)
        console.log(e)
        console.log(f)
        console.log('--------------')
        g = 10
        var h = 11
    }
}

//console.log(a) // 不可见
//console.log(b) // 不可见
//console.log(c) // 不可见吗？

//console.log(d) // 块作用域使用let，不可见;但是块外的d可见
console.log(e) // 块作用域使用var，可见
console.log(f) // 块作用域隐式声明，可见
console.log(g) // 可见
console.log(h) // 可见
```

流程控制

条件分支

```
if (cond1){

}
else if (cond2) {

}
else if (cond3) {

}
else {

}
```

```
条件的False等效
    false
    undefined
    null
    0
    NaN
    空字符串

其它值都将被视为True
```

switch...case分支语句

```
switch (expression) {
   case label_1:
      statements_1
      [break;]
   case label_2:
      statements_2
      [break;]
   ...
   default:
      statements_def
      [break;]
}
```

这里最大的问题，就是穿透问题，一定要在case中恰当的使用break语句，否则就会继续顺序向下执行。

```
let x = 5 // 换成1试一试
switch (x) {
    case 0:
        console.log('zero')
        break;
    case 1:
        console.log('one');
    case 2:
        console.log('two');
    case 3:
        console.log('three');
        break;
    case 5:
    case 4:
```

```
        console.log('four');
    default:
        console.log('other')
        // break;
}
```

switch...case语句都可以写成多分支结构。

for循环

```
// C风格for循环
for ([initialExpression]; [condition]; [incrementExpression])
{
   statement
}
```

```
for (let i=0;i<10;i++){
    console.log(i)
}
console.log('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')

for(var x=0,y=9;x<10;x++,y--){
    console.log(x*y)
}
console.log('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')

for (let i=0;i<10;i+=3){ // 步长
    console.log(i) // i++ ++i
}
```

while循环 和 do...while循环

```
while (condition)
   statement
```

条件满足，进入循环，条件为真，继续循环

```
do
   statement
while (condition);
```

先进入循环，然后判断，为真就继续循环

```
let x = 10;
while (x--) {
    console.log(x);
}
console.log('~~~~~~~~~~~~~~~~')
do {
    console.log(x);
}while(x++<10)
// 分析这个程序的打印结果
```

练习

九九乘法表，使用JS实现

```
for (let i=1;i<10;i++){
    line = '';
    for (let j=1;j<=i;j++)
        line += `${j}*${i}=${i*j} `;
    console.log(line)
}
```

for...in循环

对象操作语句for...in用来遍历对象的属性

```
for (variable in object) {
   statements
}
```

```
// 数组
let arr = [10, 20, 30, 40];

console.log(arr[1]) // 20

for (let x in arr)
    console.log(x); // 返回索引

for (let index in arr)
    console.log(`${index} : ${arr[index]}`); //插值

// C风格
for(let i=0;i<arr.length;i++)
    console.log(arr[i]);

// 对象
let obj = {
    a:1,
    b:'magedu',
    c:true
};

console.log(obj.a);
console.log(obj['b']); // 对象属性当索引访问
console.log(obj.d);  // undefined
console.log('~~~~~')

for (let x in obj)
    console.log(x); // 属性名

for (let key in obj) // 返回属性名，如同key
    console.log(`${key} : ${obj[key]}`);
```

for in 循环返回的是索引或者key，需要间接访问到值。
数组反正返回的是索引，C风格for循环操作可能方便点。根据个人喜好选择。
对象用for in合适。

for...of 循环

ES6的新语法

```
// for of
let arr = [1,2,3,4,5]
let obj = {
    a:1,
    b:'magedu',
    c:true
}


for (let i of arr) { // 返回数组的元素
    console.log(i)
}

for (let i of obj) {// 异常，不可以迭代
    console.log(i)
}
```

注意：for ... of 不能迭代一个普通对象。
原因是，of后面必须是一个迭代器（TypeError: obj[Symbol.iterator] is not a function）
可类比python中的for in，例如for x in [ ]

break 、 continue

break 结束当前循环
continue 中断当前循环，直接进入下一次循环

for迭代的差别

```
function sum(arr){
    for (let x in arr){ // 遍历index或对象属性
        console.log(x, typeof(x), arr[x]);
    }
    for (let x of arr){ // 遍历元素
        console.log(x, typeof(x));
    }
    for (let x=0;x<arr.length;x++){ // 自己定义索引数值遍历
        console.log(x, typeof(x), arr[x]);
    }
}

sum([3,6,9]);
```