正常情况下js的代码从上往下执行，通过流程控制之后可以不遵循这个规则

条件判断

```
单向分支：如果符合条件则执行，不符合则跳过
if(条件){

}

双向分支
if(条件){
 条件成立时执行
}else{
  条件不成立时执行
 }

 多向分支
 if(条件){
  成立时执行
 }else if(条件){
  成立第二个条件时执行
 }else{
  条件均不成立时执行
 }
```

switch...case

```
switch(条件){
 case 分支:
 break;
 case 分支:
 break;
 //当所有的case都不满足时执行 一般放置在所有的case后面
 default:
 break;
}
```

循环

for循环

```
for(初始值;判断条件;改变条件){
 循环体
}

常见的例子：
for(var i=0;i<5;i++){
 console.log(i)
}
```

循环中的关键字

break:终止整个for循环，后面的内容将不会执行

```
for(var i=1;i<10;i++){
 // 如果满足条件则执行,不满足则跳过
 if(i%2==0){
 // 跳出整个循环体(跳出整个for循环，for循环的代码到此结束)
 break;
 }
 console.log(i)//1
}
```

continue：结束当前循环，继续执行下一个循环

```
for(var i=1;i<10;i++){
 // 如果满足条件则执行,不满足则跳过
 if(i%2==0){
 //跳出本次循环，继续执行下一个循环
 continue;
 }
 console.log(i)//1 3 5 7 9
 }
```

while循环

```
while(判断条件){
 循环体
}

示例：
var i=0;
while(i<=10){
```

```
  console.log(i);
  i+=2;
 }
```

do...while循环

```
do{
  循环体
}while(判断条件)

var num = 5;
do{
  console.log(num)
}while(num>10)
```

注意：

while是先判断再执行循环体内容

do...while是先执行一次循环体,再进行判断

预习：函数

1、函数如何定义和调用，不同的定义方式有啥区别

2、函数的名字是否可以重复

3、匿名函数

4、函数的参数(形参、实参)

5、函数的返回值

拓展作业：

1、日历（给两个变量，一个为年(year)，一个为月(month)）

功能：输入对应的年和月之后，输出那一年当前个月份的天数

提示：1 3 5 7 8 10 12月固定31天

4 6 9 11月固定30天

2月份（分为平年和闰年　平年28天，闰年29天）

使用技术：switch...case..　　if...else...

显示的语句：2020年的2月份有29天（涉及到变量和字符串拼接）

2、9*9的乘法表（统一使用for循环来写）

函数

函数的概念：函数是一小块代码的封装，要求功能越独立越好，功能越强大越好。

函数的定义：

1、标准定义方式

```
function 函数名(参数){
 函数体
}

function fun(){
 console.log('标准函数定义')
}
```

注意：函数可以在任意的地方调用

2、赋值的方式

```
var 函数名 = function(参数){
 函数体
}

var fun = function(){
 console.log('通过赋值方式定义')
}
```

注意：赋值方式定义的函数只能在赋值之后才能调用

3、原生方法定义（了解,不建议使用）

```
var 函数名 = new Function(函数的具体内容);

var fun = new Function('console.log("原生方法定义")');
```

函数的调用:将函数体内的代码从上往下执行一遍

```
函数名(); 例如：fun() demo(实参)
```

函数重名问题

当在一个js中，标准方式定义的函数名相同时，后面的函数会自动覆盖前面定义的函数，在任何地方都没法调用

注意：函数命名时，不能和系统函数重名

匿名函数

匿名函数：没有名字的函数

匿名函数的特点：

1、避免了函数取名的困扰

2、匿名函数永远不会被其他函数重写

3、匿名一般情况下，只调用一次。定义时则直接调用

函数的参数

函数的参数

形参----------形式参数(定义函数时小括号内的参数为形参)

实参----------实际参数(调用函数时，小括内的参数为实参)

```
function 函数名(形参1，形参2...){
 函数体...
}

函数名(实参1,实参2...)；//函数调用
```

注意：形参和实参的个数不一定要相等

形参和实参的个数问题?

1、函数给出形参，但为传入实参

```
function fun1(a){
 console.log(a)//undefined
}
fun1();
```

2、函数给出的形参个数大于实参个数

```
function fun2(a,b,c){
 console.log(a,b,c)//1,2,undefined
}
// 实参按照从左往右的顺序分别给形参赋值
fun2(1,2)
```

3、函数给出的形参个数等于实参个数（常规情况）

```
function fun3(a,b){
 console.log(a,b)//10 20
}
fun3(10,20);
```

4、函数给出的形参个数小于实参个数

```
function fun4(a){
 console.log(a)//1
}
fun4(1,2,3)//a=1 2 3则没有参数来接收
```

5、函数给出实参，但未给出形参

```
function fun5(){
 // 可以通过arguments接收所有的参数,采用数组的方式返回
 console.log(arguments)//["宋鑫", "禽类", "静静"]

 // 数组的下标默认从0开始,可通过指定下标获取对应的值
 console.log(arguments[2])//静静
}
fun5('宋鑫','禽类','静静')
```

注意：系统有一个arguments可以接收所有的实参

函数的返回值return

注意：console.log()只是输出，函数的返回值只和return有关

函数的返回值的几种情况

1、函数调用后未给出return返回值,则调用后的结果为undefined

```
function fun1(){
 console.log('测试代码：不给返回值')

 // 此时函数调用后仍然为undefined.因为没有返回具体的值
```

```
 return
}
// 将函数调用的结果赋值给一个变量,该变量就就接收函数return的返回值
// 如果没有return,则返回undefined
var res = fun1()
console.log(res)//undefined
```

2、函数调用后通过return返回值（可以为任意数据类型，变量，表达式）（重点）

函数调用的结果则为函数返回值（return后面的内容）

```
function fun2(){
 var name = '禽类';

 var obj = {name:'俊哥',age:30}
 // return 111;
 // return '返回字符串';
 // return true;

 // 返回一个变量
  // return name;//禽类
  // return obj;//{name: "俊哥", age: 30}
  // return {
  // name:'曹兵',
  // age:18
  // }//{name: "曹兵", age: 18}
  return 5+2//7
}
var res = fun2()
console.log(res)//111
```

3、return后面的代码不执行

```
function fun3(){
 console.log('我不想看到你')
 return 111;

 // return后面的代码不会被执行
 console.log('看看我在不在')
}

```

```
var res = fun3();
console.log(res)
```

4、函数调用后可以给出返回值，也可以不给（return不是必须的）

函数的变量

作用域：变量的作用范围（使用的范围的大小）

全局变量：

定义：在函数体的外部定义的变量（推荐），或者在函数体内部不采用var声明的变量（但是不推荐此写法）

作用域：作用范围广，可以在任意地方均可以访问和修改，故叫做全局变量。只有整个js文件运行结束后，全局变量才销毁。

注意：一旦全局变量被更改，所有使用的地方都会被影响。所以合理使用。

局部变量：

定义：在函数体的内部采用var定义的变量

作用域：作用范围只在函数体的内部，作用范围小，故叫做局部变量。函数执行完成后，局部变量则被销毁

注意：

建议能使用局部变量则尽量使用局部变量

不同的函数之间，局部变量名可以重复使用
当全局变量和局部变量发生冲突时，优先使用局部变量

函数的作用域链

函数的作用域链：变量是（从内往外）一级一级的向上查找（父级函数），直到找到为止，反之不成立

函数查找变量的方式：
1、先查找当前函数是否具有该变量，则使用该变量
2、若当前函数没有，则向父级函数查找是否有改变量，若有，则使用改变量
3、若直到找到最大的作用域，没发现该变量，则系统认为该该变量不存在

拓展作业：

1、封装计算器函数

功能：必须实现 + - * / 四种基本运算
参数：三个(两个数值和一个运算方法)
结果通过返回值返回

形式：通过传递不同的实参，得到不同的计算结果

```
// 封装计算器
// 方法一：
function calculator(a,b,c){
```

```
switch(c){
 case '+':
 // return可以返回表达式
 return a+b;
 break;
 case '-':
  return a-b;
  break;
  case '*':
  return a*b;
  break;
  case '/':
  return a/b;
  break;
  default:
  return '该运算我还不会哟！'
  break;
}

var res = calculator(3,8,'*')
console.log(res)
```

2、观察以下数字1 2 4 8 16 32 64 128 256 512.....，封装一个函数，返回第10个数字的结果

1*2*2*2*2=16

```

// 方法一：
var res = 1;
function fun1(n){
 // console.log(3)

 for(var i=1;i<n;i++){
 res*=2;//res = res*2;
 }

  // 将res得到的结果返回
```

```
 return res;
}

var result = fun1(10)//512
console.log(result)


// 方法二:
// 新的方法(涉及到数学函数)
function fun2(n){
 // pow(底数,指数)
 return Math.pow(2,n-1)
}

var result2 = fun2(10);
console.log(result2)
```

回调函数（重点）

定义：形参中至少有一个是已定义好的函数

回调的意义：我们可以在函数的内部调用其他函数（且函数名还可以切换）

```
// 加
function add(a,b){
 return a+b;
}

// 减法
function sub(a,b){
 return a-b;
}

// 乘法
function mul(a,b){
 return a*b;
}
```

```
15
16 // 除法
17 function divi(a,b){
18  return a/b;
19 }
20
21 // 核心代码
22 function calculator(a,b,method){
23  console.log(method)
24
25  // method是函数名 函数名()不就是函数的调用
26  // 此时的method就是sub函数 method(a,b)
27  return method(a,b);//add(a,b)
28 }
29 // sub代表的是函数名 不能随意加上引号 不能加小括号
30 var res = calculator(4,5,add)
31 console.log(res)
```

闭包函数（重点：注意笔试面试考闭包的概念）

闭包函数:内部函数在其作用域外被调用,则形成了内部函数的闭包

若闭包函数使用了全局变量，则该变量所占的内存不会被释放。直到整个js运行结束。

闭包函数的特点：

优点：内部的函数可以在其作用域之外被访问

缺点：占内存，可能存在内存泄漏，慎用闭包。除非必须使用某个局部变量

解决方案：在内部函数调用完成之后，将该内存释放。(将该变量赋值为空)

// 闭包:内部函数在其作用域外被调用,则形成了内部函数的闭包

```
function outside(){
 // function inside(){
 // console.log('sss')
 // }

 //系统会认为已存在num变量,后面几次调用不会重新覆盖
 var num = 100;
 var inside = function(){
 num++;
  console.log(num)
  }

  // 将整个函数作为返回值返回
  return inside;

}

var res = outside()
console.log(res)//此时的res 接收的是Outside返回的函数

// 调用返回的函数
//普通调用方式，该方式会造成内存占用，可能存在内存泄漏
res()//101
outside()
res();//102
outside()
res();//103


//该调用方式，在调用结束后手动释放内存
var res = outside()
res()//101
```

```
33
34 // 将返回值返回的函数进行清空,释放内存空间
35 res = null;
```

递归函数（考试方式:封装一个函数）

定义：自己调用自己

```
1 function fun(n){
2  console.log(n)//3 2 1 0
3  if(n>0){
4  fun(n-1)//fun(2) fun(1) fun(0)
5  }
6 }//3 2 1 0
7
8 function fun(n){
9  console.log(n)//3 2 1 0
10  // 在递归函数中,若在自己调用自己后还有代码,则该代码需要等到不满足条件时,依次返回执行
11  if(n>0){
12  fun(n-1)//fun(2) fun(1) fun(0)
13  }
14  console.log(n)
15 }
16 fun(3)//3 2 1 0 0 1 2 3
17
```

递归实现阶乘（笔试题）

```
1 var num = 1;
2 function fun(n){
3 if(n>1){
4  num*=n;//num = num*n; 1*3*2*1
5  fun(n-1)//fun(2) fun(1)
6 }else{
7  return 1;
8 }
9  // 将阶乘的结果进行返回
10  return num;
11 }
12
13 var res = fun(3);//1*2*3*4...*10
14 console.log(res)
```