目录

今日作业

作业讲评

```
<script type="text/javascript">
    // 1.求一组数值 var arr=[10,4,20,15,6,9]  的最小值
    var arr = [10,4,20,15,6,9];
    // 假设数组第一个元素是最小值,放到minValue变量中
    var minValue = arr[0];
    for( var i=1; i<arr.length; i++){
        if( minValue > arr[i] ){ // 如果数组元素比我假设的值更小
            // 我们就把对应的数组元素让他成为新的最小值
            minValue = arr[i];
        }
    }
    document.write("最小值为:" + minValue);
</script>
```

```
<script type="text/javascript">
    // 2.var arr= [2,6,1,77,52,25,7] 然后将
    这个数组中的元素为奇数的平均值求出来
    var arr = [2,6,1,77,52,25,7];
    var sum = 0;// 统计奇数的和
    var count = 0;// 统计奇数的个数

    for(var i = 0;i< arr.length; i++){
        if( arr[i]%2 == 1){//
        判断数组元素是否为奇数
            sum = sum + arr[i];
            count++;
        }
    }
    // 奇数平均值 = 奇数的和/奇数的个数
    console.log( "平均值为:" + sum/count );
</script>
```

```
var arr = [
    ['小A','女',21,'PHP'],['小B','男',23,'C'],
    ['小C','男',24,'C'],['小D','女',21,'PHP'],
    ['小E','女',22,'C'],['小F','男',21,'PHP'],
    ['小G','女',22,'JAVA'],['小H','女',20,'JAVA'],
    ['小I','女',20,'PHP'],['小J','男',20,'PHP']
];

var str = "<table border='1' rules='all' width='400'>";
    str += "<tr>";
    str += "<th>姓名</th><th>性别</th><th>年龄</th><th>学科</th>";
    str += "</tr>";
    for(var i=0;i<arr.length;i++){
        str += "<tr>";
        for(var j=0;j<arr[i].length;j++){
            str += "<td align='center'>"+arr[i][j]+"</td>";
        }
        str += "</tr>";
    }
    str += "</table>";

document.write( str );
```

```
// 4.将数组  var arr = [2, 3, 6, 1, 77, 8, 52, 9, 25, 7]  大于等于 10 的 元素选出来 , 放入 新数组,
新数组结果如下 [77,52,25]

var arr = [2, 3, 6, 1, 77, 8, 52, 9, 25, 7];
// 定义一个新空数组
var newArr = [];

for( var i=0;i<arr.length; i++){
   if( arr[i] >= 10 ){// 如果数组元素大于等于10

       // 往数组中添加元素
       newArr[ newArr.length ] =  arr[i] ;
   }
}
console.log( newArr );
```

学习目标

- 能够掌握和理解函数的概念和应用
- 能够定义函数
- 能够调用函数
- 能够理解函数的参数
- 能够理解函数中return的作用
- 能够使用匿名函数
- 能够了解变量的作用域
- 能够区别值传递与引用传递

函数

为什么需要函数?

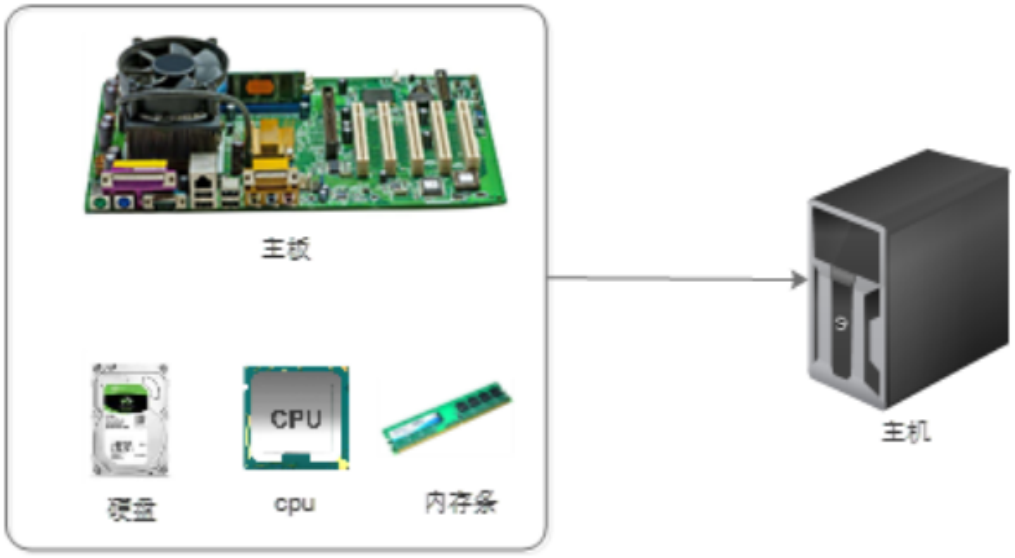
虽然for也能实现一些简单的 重复操作，但是 比较具有局限性。

我们js 里面，也有非常多的相同代码，可能需要大量重复使用。 此时我们可以利用函数。

- **为了解决代码的重用!**
- **减少代码量。**

函数的封装

- 函数的封装是把一个或者多个功能通过函数的方式封装起来，对外只提供一个简单的函数接口
- 简单理解：封装类似于将电脑配件整合组装到机箱中 (类似快递打包)



什么是函数?

函数 就是 封装了 一段 可被**重复调用执行**的 代码块。 可以实现大量代码的重复使用。

函数是可以被命名的，它是为了解决某些功能的代码段!

可以被命名：表示函数是可以有名字的，也是可以没有名字的。

代码段：函数体

函数的分类

分为系统内置函数和自定义函数

系统内置函数就是js自带的函数,我们直接调用即可,比如document.write, console.log

自定义函数就是程序员自己封装的函数

定义(声明)函数

语法:

```
function 函数名([参数列表]){
  这里放函数体,函数体就是要重复执行代码块
}
```

结构说明:

- function它是定义函数的关键字 **不可以省略。**
- **函数名它的命名规则与变量名是一样的**
  - 可以使用大小写英文字母、下划线(_)、美元符号($)、数字组成
  - 不能使用数字开头
  - 不能使用JS中的关键字和保留字作为函数名
  - 如果函数是由多个单词组成的，那么可以使用下划线连接法和驼峰法
  - 严格区分大小写
- 函数名后面紧跟着**一对小括号 这一对小括号不能省略**
- 小括号里面可能有**参数**，我们将其称之为**形参**
- 小括号后面紧跟着**一对大括号 这一对大括号不能省略**
- 大括号里面是**函数体**

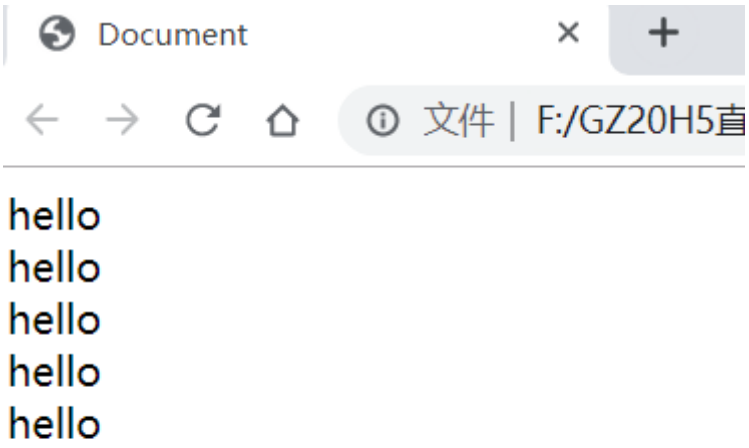
注意： 声明函数本身并不会执行代码，只有调用函数 时 才会执行函数体

举例: 定义一个sayHello的函数

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<title>Document</title>
<script type="text/javascript">
/*
定义函数的完整语法如下
function 函数名([形参1,形参2,形参3...]){
 函数体;
 return 返回值;
 }

 结构说明:
 1. function是关键字,不能改,不能省略
 2. 函数名可以自定义,函数名的规则跟规范跟变量名一致
 3. 函数名后面()里面放的形式参数列表,形参个数可以有多个,也可以一个都没有,也就是省略形参;形参的作用是用于给函数传值
 4. 函数体就是{}里面的代码块,我们就叫函数体,函数体只会"函数被调用"的时候才会执行
 5. return关键字,return作用是向函数的调用者返回对应的值,函数如果没有return返回值,调用者会得到一个undefind
 */


 // 定义函数的步骤:重点
 // 第一步:确定需要重复使用代码(如果有不确定值的时候,就需要使用形参),这个就是函数体
 // 第二步:定义函数,用function把函数体包起来
 // 第三步:确定是否需要返回值

 // 举例: 定义一个sayHello的函数
 function sayHello(){
 document.write("hello<br/>");
 }
 // 定义函数,不会执行函数中函数体的代码

 /*
 调用函数的语法
 函数名([实参1,实参2,实参3...]);

 说明:
 1. 如果函数定义的时候,有形参,那么调用函数的时候,就需要传递实参
 2. 如果函数定义的时候,没有形参,那么调用函数的时候,就不需要传递实参
 3. 调用几次函数,对应的函数体就会被执行几次
```

```
42  */
43
44  // 调用函数
45  sayHello();
46  sayHello();
47  sayHello();
48  sayHello();
49  sayHello();
50  </script>
51 </head>
52 <body>
53
54 </body>
55 </html>
```



调用函数

格式:

```
1  函数名([实参列表])
```

注意: 在定义函数时如果有形参 反过来说 在调用的时候也要有实参 但是这个不是绝对的!

注意:函数一定是先定义后再调用

补充说明:虽然JS语言它有预解析的功能: JS它会将变量的定义与函数的定义提升到当前的script标签的最前面 但是还是建议函数先定义后调用

练习: 写一个函数计算1~100之间所有数的和

```
<script type="text/javascript">
    // 练习: 写一个函数计算1~100之间所有数的和

    // 定义函数的步骤:重点
    // 第一步:确定需要重复使用代码(如果有不确定值的时候,就需要使用形参),这个就是函数体
    // 第二步:定义函数,用function把函数体包起来
    // 第三步:确定是否需要返回值

    // 定义函数,定义函数的时候,函数体不会被执行
    function getSum(){
        var sum = 0;
        for(var i=1;i<=100;i++){
            sum += i;
        }
        document.write( sum );
    }

    // 调用函数
    getSum();
</script>
```

同名函数后面的会将前面的覆盖

举例:

```
    <meta charset="UTF-8">
    <title>Document</title>
    <script type="text/javascript">
        // js有预解析的功能,函数会被提前声明定义

        // 定义多个相同名字的函数,后面定义的,
        会覆盖前面定义的
        function fn(){
            document.write("我是第一个函数<br/>");
        }
        fn();

        // 我们又定义了跟fn同名的函数
        function fn(){
            document.write("我是第二个函数<br/>");
        }
        fn();

        // 我们又定义了跟fn同名的函数
        function fn(){
            document.write("我是第三个函数<br/>");
        }
        fn();
    </script>
</head>
<body>
```

Document

文件 | F:/GZ20H5直播6班/Javascript基础/05...

我是第三个函数
我是第三个函数
我是第三个函数

函数参数

参数 ： 在函数内部某些值不能固定，只有在调用的时候才能确定这个值

参数	定义

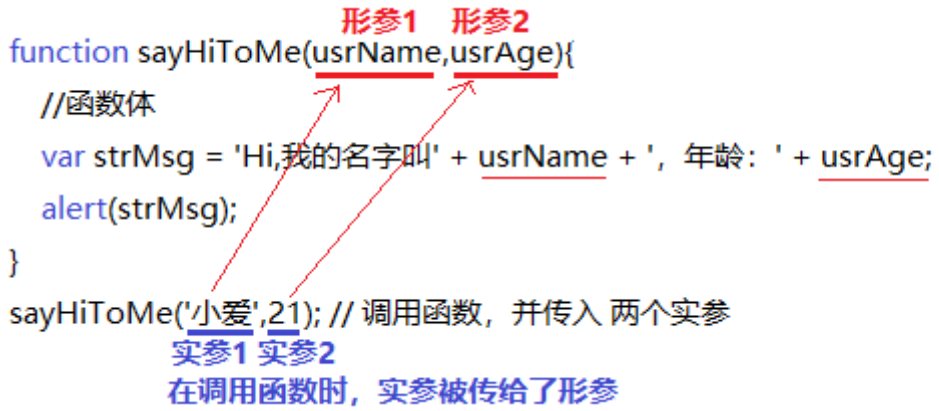
形参	**形式上的参数 函数定义的时候 传递的参数 当前并不知道是什么**
实参	**实际的参数 函数调用的时候传递的参数 这个参数我们知道 给形参赋值**

注意：在一个函数中，参数的多少是根据功能来定义的！

语法：

```
// 带参数的函数声明
function 函数名(形参1, 形参2 , 形参3...) { // 可以定义任意多的参数，用 逗号 分隔 形参前不要加var
  // 函数体
}

// 带参数的函数调用
函数名(实参1, 实参2, 实参3...);
```

图解：



案例:用函数封装 根据传入的 任意2个数值的和

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script type="text/javascript">
  // 我们在定义函数的时候,如果函数体中有一些不确定变量的值,或者函数体中有些需要调用的时候才能确定值,这样的值,我们就会使用函数的参数的实现

  // 函数的参数分为两种,定义函数的时候,叫形式参数,简称"形参"
   // 函数的参数分为两种,调用函数的时候,叫实际参数,简称"实参"
   // 其实形式参数就是一个变量名,注意,不能加var


```

```
/*函数参数的使用语法

function 函数名([形参1,形参2,形参3...]){
函数体;
return 返回值;
}

函数名([实参1,实参2,实参3...]);*/

// 实现一个看到谁,就对谁说你好的函数
function fn( username ){
// document.write( username );
// document.write("<br/>");
document.write( username+ ",你好<br/>");
}

// 有形参的时候,对应调用函数的时候,就要传递一个值给形参;如果调用的时候,没有传实参,对应形参的是就是undefind

// 不传实参
fn();
// 传实参
fn( "张三" );
fn( "李四" );

document.write("<hr/>");




// 案例:用函数封装 根据传入的 任意2个数值的和
function getSum(a,b){
document.write( a + b);
document.write("<hr/>");
}

getSum(10,20);
getSum(50,30);
</script>

```

```
</head>
<body>

</body>
</html>
```

```
// 案例:用函数封装 根据传入的
任意2个数值的和
function getSum(a,b){
    document.write( a + b);
    document.write("<hr/>");
}

getSum(10,20);
getSum(50,30);
```

30

80

练习:定义一个函数,可以传入任意3个不相等数值,然后输出值最大的那个数,比如10,20,30输出30

```
<script type="text/javascript">
 // 第一种方法
 function getMax1(a,b,c){
 var arr = [a,b,c];
 var maxValue = a;
 for(var i=1;i<arr.length;i++){
 if(maxValue < arr[i] ){
 maxValue = arr[i];
 }
  }
  document.write( maxValue );
  }

  getMax1( 10,50,20 );

  document.write("<hr/>");

  // 第二种方法
  function getMax2(a,b,c){
  // 基本上可以使用if...else双分支结构的,都可以使用三目运算符

  // 三目运算符的语法
```

```
// 条件表达式 ? 成立时的代码 : 失败时的代码;

// 先找到a和b中大的那个值
var max = a > b ? a : b;
// 再拿a和b中大的那个值跟另一个比
max = max > c ? max : c;
document.write( max );
}
getMax2(5,1,75);

document.write("<hr/>");

// 第三种方法
function getMax3(a,b,c){
var max = (a > b ? a : b) > c ? (a > b ? a : b) : c;
document.write( max );
}
getMax3(5,1,75);
document.write("<hr/>");

// 第四种方法
function getMax4(a,b,c){
var max = a > b ? (a>c ? a:c) : (b > c ? b:c);
document.write( max );
}
getMax4(5,1,75);
</script>
```

函数参数总结:

- 函数可以带参数也可以不带参数
- 声明函数的时候，函数名括号里面的是 形参 类似一个变量 接受数据的
- 调用函数的时候，函数名括号里面的是 实参。
- 多个参数中间用逗号分隔。

- 形参的个数可以和实参个数不匹配，但是结果不可预计，我们尽量要匹配。

函数形参和实参数量不匹配时

函数调用的时候实参不是必需的,javascript允许在调用函数的时候省略实参

```
function fn(a,b){
   console.log( b );
}
fn(1,2,3);
fn(1);
fn();
```

通过函数名.length可以得到形参的个数

```
function fn(a,b){
   console.log( b );
}
console.log( fn.length );
```

不能省略前面的参数,只能省略后面的参数;如果一定要省略前面的参数,可以传入 `undefined`

```
function fn(a,b){
   console.log( b );
}
// fn(,1)// 错误
fn(undefined,1)
```

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // 函数形参和实参数量不匹配时
 function fn(a,b,c){
 console.log(a,b,c);
```

```
10  }
11
12  // 函数调用的时候实参不是必需的,javascript允许在调用函数的时候省略实参
13
14  // 实参个数少于形参个数
15  fn(10);// 10 undefined undefined
16  fn(10,20);// 10 20 undefined
17
18  // 实参个数等于形参个数
19  fn(10,20,30);// 10 20 30
20
21  // 实参个数多余形参个数
22  fn(10,20,30,40,50,60);// 10 20 30
23
24  console.log("");
25  // 我们通过 函数名.length 可以得到形参的个数
26  console.log("fn函数的形参个数为:" + fn.length );
27
28  // 不能省略前面的参数,只能省略后面的参数;如果一定要省略前面的参数,可以传入unde
fined
29  // fn( ,20,30);// 会报错
30
31  // 如果一定要省略前面的参数,可以传入undefined
32  fn( undefined , 20 , 30);
33  </script>
34 </head>
35 <body>
36
37 </body>
38 </html>
```

函数参数的默认值

语法: 需要定义默认值的参数,需要放在形参列表的后面

```
1 function 函数名(形参1,形参2=默认值2,形参3=默认值3...){
2     函数体;
3 }
```

举例:

```html
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // 我们可以在定义函数的时候,给形参设置默认值;
 // 如果有形参有默认的话,调用函数的时候,可以不传递实参;不传实参就会使用默认值
 // 注意:需要定义默认值的参数,需要放在形参列表的后面


 // 定义函数,函数参数没有设置默认值
 function fn(a,b,c){
 console.log(a+b+c);
 }
 fn(10,20,30);// 60
 fn(10,20);// NaN
 console.log("");



 // 注意:需要定义默认值的参数,需要放在形参列表的后面
 function fn2(a = 10,b = 20,c){
 console.log( a+b+c );
 }
 fn2(50,7,12);// 69
 fn2(7,12);// NaN
 console.log("");



 // 设置了默认值的形式参数,需要形参列表的后面
 // 如果有形参有默认的话,调用函数的时候,可以不传递实参;不传实参就会使用默认值
 function fn3(a,b=20,c=10){
 console.log( a+b+c );
 }
 // a,b,c三个都传值了
 fn3(10,50,60);// 120

 // a,b两个传值了
 fn3(10,40);// 60
```

```
42
43  // 只传了一个a的值
44  fn3(10);// 40
45  </script>
46 </head>
47 <body>
48
49 </body>
50 </html>
```

return关键字 难点,重点

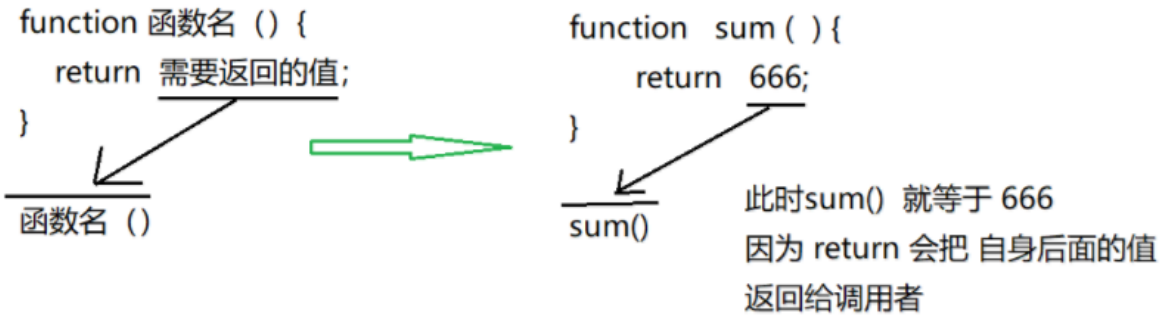
一般在函数体里面不会有输出语句，**只会有一个return关键字，将我们要输出的内容返回给函数的调用者。**

- **return在英文中表示"返回"的意思**
- **return关键字是在函数体里面使用**
- **如果函数体中没有return值,那么函数的调用者得到的结果是undefind**

语法:

```
1 function 函数名(){
2     return   需要返回的值;
3 }
4
5 //函数调用
6 函数名())   //    此时 调用 函数就可以得到    函数体内return 后面的值
```

图解:



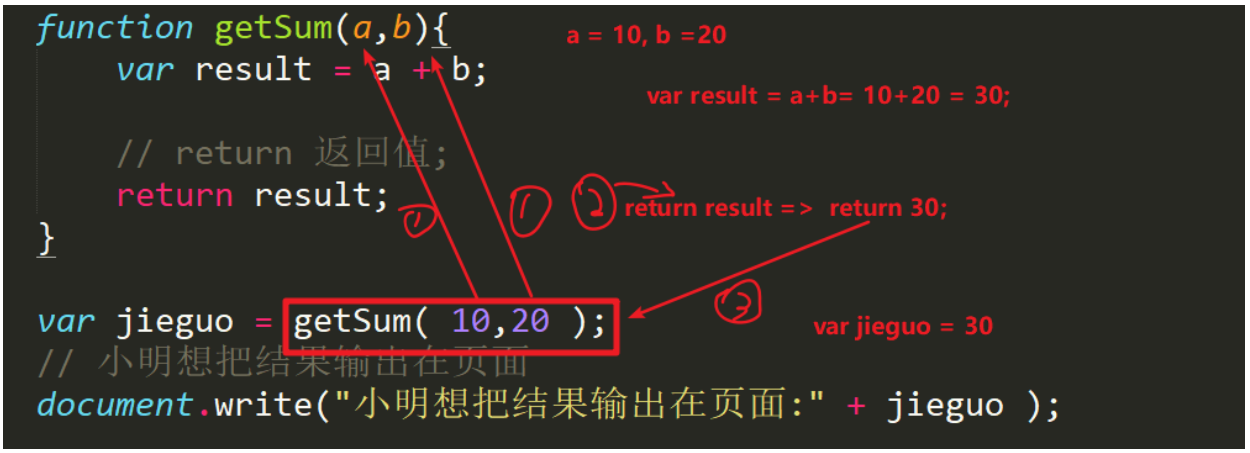
举例:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
```

```
<meta charset="UTF-8">
<title>Document</title>
<script type="text/javascript">

/*// 得到任意两个数的和
function getSum(a,b){
 // 直接在函数里面输出
 var result = a + b;
 document.write("他们的和为:" + result +"<br/>" );
 }

 // 小明想把结果输出在页面
 getSum(10,20);

 // 小红想把结果输出在控制台
 getSum(50,60);

 // 我们这个时候,发现函数体中输出只能写一个,不能同时实现两种,所以这个时候,可以考虑使用return返回值*/


 // return关键字可以在函数中向函数的调用者返回结果
 // 函数外面需要向函数里面传值,我们会通过函数的参数来实现
 // 函数里面需要向函数外面传值,我们会通过函数返回值实现


 /*return使用的语法,在函数体中写,return后面的值就会被返回给"函数调用者"
 return 返回值;*/
 // 特别要注意:return没有输出的功能
 function getSum(a,b){
 var result = a + b;

 // return 返回值;
 return result;
 }

 var jieguo = getSum( 10,20 );
 // 小明想把结果输出在页面
 document.write("小明想把结果输出在页面:" + jieguo );

 // 小红想把结果输出在控制台
```

```
44   var jieguo2 = getSum(50,60);
45   console.log("小红想把结果输出在控制台" + jieguo2 );
46
47
48   console.log( getSum(100,150)/10 );// 25
49   </script>
50 </head>
51 <body>
52
53 </body>
54 </html>
```

举例代码图解:



举例: 如果函数体中没有return值,那么函数的调用者得到的结果是"undefind"

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script type="text/javascript">
        // 如果函数体中没有return值,那么函数的调用者得到的结果是"undefind"
        // return了什么,函数的调用者就得到什么
        function fn(a,b){
            var result = a+b;
        }

        var jieguo = fn(10,20);
        console.log( jieguo );// undefined,因为函数体中没有return返回东西

        console.log("");

        function fn2(a,b){
            var result = a+b;
            return 100;
        }
        var jieguo2 = fn2(10,20);
        console.log( jieguo2 );// 100,因为函数体中返回的是100
    </script>
</head>
<body>
```

return它在函数体使用有两层含义:

第一层含义: 当函数体里面遇到了return关键字以后，当前的这个函数就不会再往下进行执行了。

第二层含义:它会向函数的调用者返回数据 (重点) 返回值

```
格式:
 return 数据;

在调用函数时 可以定义一个变量要接收到这个函数的返回值
```

举例:

```html
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">

 /*function fn1(){
 console.log(1111);
  console.log(22222);
  console.log(333333);
```

```
12  }
13
14  // 调用函数,就要执行函数体
15  console.log( fn1() );//会输出 1111 22222 333333 undefind*/
16
17
18  // return它在函数体使用有两层含义:
19  // 第一层含义: 当函数体里面遇到了return关键字以后，当前的这个函数就不会再往下进行执行了。
20  /*function fn2(){
21  console.log(1111);
22  console.log(22222);
23  return "我是函数中的返回值";
24  console.log(333333);
25  }
26
27  document.write( fn2() );*/
28
29
30  /*function fn3(){
31  console.log(1111);
32  return 666666;
33  console.log(22222);
34  return "我是函数中的返回值";
35  console.log(333333);
36  }
37  fn3();*/
38
39  // 第二层含义:它会向函数的调用者返回数据 (重点) 返回值,也就是return了什么,函数调用者就得到什么
40  function fn4(){
41  return "abc";
42  return "456";
43  return "zhangsan";
44  }
45  console.log( fn4() );// abc
46  </script>
47 </head>
48 <body>
49
50 </body>
```

```
51 </html>
```

练习:定义一个函数求一个任意数组,数组元素的最大值 ,并返回在这个最大值 , 比如 [5,2,99,101,67,77] 中元素最大值是101,返回101

```
// 不固定的东西,需要放在函数的参数中
function getArrayMax( arr ){
    var maxValue = arr[0];
    for(var i=1;i<arr.length;i++){
        if(maxValue < arr[i] ){
            maxValue = arr[i];
        }
    }
    // 返回最大值
    return maxValue;
}

var arr1 = [5,2,99,101,67,77];
console.log( getArrayMax( arr1 ) );

var arr2 = [3,8,2,99,1,88];
console.log( getArrayMax( arr2 ) );
```

注意：我们在以后的工作中，函数体里面尽量不要有输出语句(document.write alert console.log)，尽量的使用return关键字将数据返回给函数的调用者。

特别强调：在一个函数里面，**return关键字可以有多个，但是return只会执行一次**

最后再补充说明一点: return关键字它只能返回一个数据，如果需要返回多个数据，我们可以将多个数据使用数组的方式来保存，然后再将数组返回。

```
<script type="text/javascript">
    // 如果想在函数里面返回多个值,可以使用数组;因为数组就是可以保存多个值的
    /*function fn(){
        return 1;
        return 2;
        return 3;
    }
    var result = fn();
    console.log( result ); */


    function fn(){
        // 返回了一个数组
        return [1,2,3];
    }
    var result = fn();
    console.log( result );

    // 通过下标就可以从数组中找到某一个数组元素
    console.log( result[1] );
</script>
```

函数中的arguments对象 了解

当不确定有多少个参数传递的时候，可以用 arguments 来获取。JavaScript 中，arguments实际上它是当前函数的一个内置对象。所有函数都内置了一个 arguments 对象，arguments 对象中存储了传递的所有实参。arguments展示形式是一个伪数组，因此可以进行遍历。伪数组具有以下特点：

- 具有 length 属性
- 按索引方式储存数据
- 不具有数组的 push，pop 等方法

注意：在函数内部使用该对象，用此对象获取函数调用时传的实参。

举例: 利用函数求任意个数的数字中的最大值

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
```

```
// 当不确定有多少个实际参数传递的时候，可以用 arguments 来获取。JavaScript 中，arguments实际上它是当前函数的一个内置对象。所有函数都内置了一个 arguments 对象，arguments 对象中存储了传递的所有实参。arguments展示形式是一个伪数组，因此可以进行遍历。伪数组具有以下特点：

// 具有 length 属性
// 按索引方式储存数据
// 不具有数组的 push , pop 等方法
// 注意：在函数内部使用该对象，用此对象获取函数调用时传的实参。


// 举例: 利用函数求任意个数的数字中的最大值
function fn(){
// arguments是js中的一个内置对象,只有在"函数体中"才可以使用
// arguments会把函数的实参都保存起来,是一个实参列表
// arguments展示形式是一个"伪数组",具有 length 属性,可以按索引方式储存数据(可以理解为数组的下标),因为不是真正的数组,不具有数组的 push , pop 等方法
// console.log( arguments );
// console.log( arguments.length );// 可以得到实参的个数
// console.log( arguments[1] );// 找到实参列表中第二个参数

var maxValue = arguments[0];
for(var i=1;i<arguments.length;i++){
if( maxValue < arguments[i] ){
maxValue = arguments[i];
}
}
document.write("最大值为:" + maxValue );
}
fn(100,1,2,3,4,5,6,7,8,9);// 得到100

document.write("<hr/>");

fn(50,3,40,70);// 得到70

</script>
</head>
<body>

</body>
</html>
```

匿名函数　用的多,必须掌握

什么是匿名函数?

没有名字的函数 称之为匿名函数!

注意：**匿名函数在JS中使用的也比较多**

匿名函数的使用

将匿名函数赋值给**变量**或者是**事件**

将匿名函数赋值给变量，然后通过变量名加上小括号来调用匿名函数

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // 匿名函数,就是没有名字的函数

 // 直接去掉函数名,会报错 Function statements require a function name
  /*function(){
  console.log("我是一朵云~");
  }*/

  // 匿名函数不能单独使用,需要赋值给变量或者事件才能使用



  // 有名函数
  function funName(){
  console.log("我是一棵小草~");
  }
  // 输出fn
  console.log( funName );

  // 有名函数的调用 语法: 函数名();
  funName();
```

```
  console.log( "" );


  // 匿名函数赋值给变量
  var a = function(num1,num2){
  console.log("我是一朵云~");
  }
  // 此时,a是函数的定义,其实这个时候变量a就是函数
  console.log( a );
  // 如果匿名赋值给了变量,那么我们该如何调用这个匿名函数? 语法: 变量名();

  a();

  </script>
</head>
<body>

</body>
</html>
```

将匿名函数赋值给事件 那么匿名函数什么时候才会执行？它要等到事件触发了以后 匿名函数才会执行。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <button>按钮</button>

  <script type="text/javascript">
  // 可以还可以把匿名函数赋值给事件
  // 事件相关知识会在后续中讲解,现在这个事件看不懂,没有关系

  // 获取dom对象
  var objBtn = document.querySelector("button");
  // 给对象绑定事件,事件驱动程序一般就是匿名函数
  // 这个时候,匿名函数什么时候触发,事件被触发的时候,匿名函数就会被触发

  // onclick代表鼠标单击
```

```
 objBtn.onclick = function(){
 alert("点击了我~");
 }
 </script>
</body>
</html>
```

立即调用的函数表达式　了解

也可以叫自调用匿名函数，简称 IIFE（Immediately-invoked function expression）

使用场景:

只需要执行一次，不需要重复调用

在定义函数之后，就会立即调用该函数,**语法如下:**

```
(function(){ 函数体 }());
// 或者
(function(){ 函数体 })();
```

```
注意:两种写法最后的分号都是必须的。如果省略分号，遇到连着两个 IIFE，可能就会报错。
(function(){ 函数体 }())
(function(){ 函数体 }())
```

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // 立即调用的函数表达式 了解
 // 也可以叫自调用匿名函数 , 简称 IIFE ( Immediately-invoked function expression )
 // 使用场景: 只需要执行一次，不需要重复调用

```

```
11  // 在定义函数之后，就会立即调用该函数,语法如下:
12
13  /*
14  (function(){ 函数体 }());
15  // 或者
16  (function(){ 函数体 })();
17  */
18
19  // 匿名函数如果不赋值给变量,也不赋值给事件,那么就需要马上调用这个匿名函数
20
21  // 第一种方法,用()包裹匿名函数,再在后面再加一个()调用
22  (function(){
23  console.log("我是喷火娃,我可以给爷爷拔火罐~");
24  })();
25
26  // 注意:两种写法最后的分号都是必须的。如果省略分号，遇到连着两个 IIFE，可能就会报错。
27
28  // 第二种方法,先在后面加一个(),最后把用一个()再包裹一个内容
29  (function(){
30  console.log("我可以大娃,我可以变得超级大大大!");
31  }());
32  </script>
33 </head>
34 <body>
35
36 </body>
37 </html>
```

今日总结

xmind总结,自己做,要交

今日作业

因为今天是周五,周末两天,作业会稍微多一点,昨天8个,周末作业才9个,具体作业在文件夹可以看到