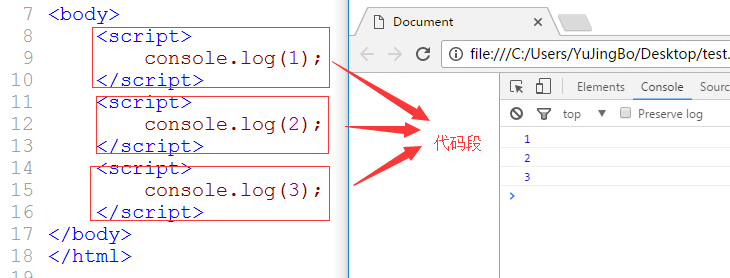
JS高级特性

# 一，JS代码的解析与执行

## 1，什么是代码段？

由script标签组成的一段代码就是代码段。

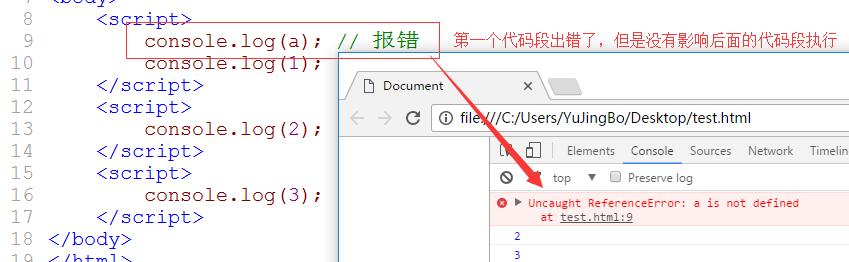


## 2，代码段都有哪些特点？

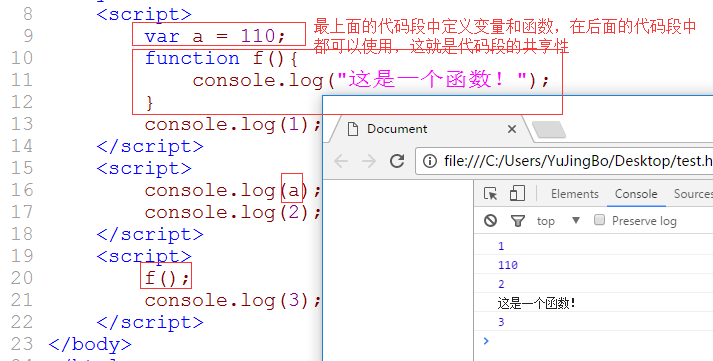
主要有两个特点：

* 1，独立
* 2，共享

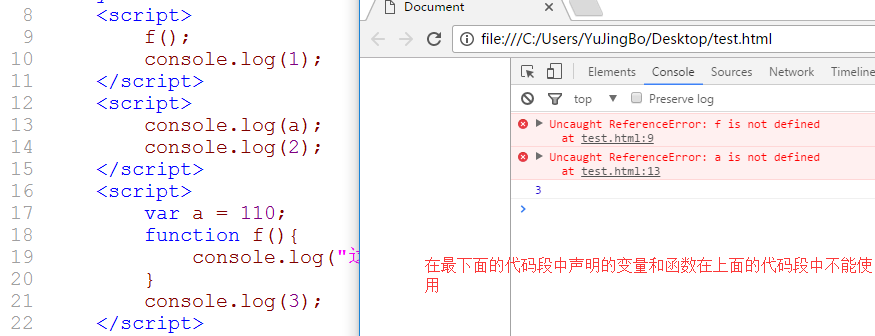
**独立：**



**共享：**

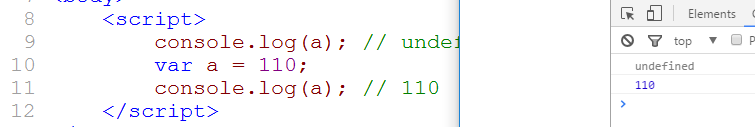


注意代码是从上到下执行的：



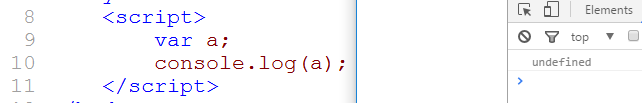
## 3，什么是预编译？

**先测试一下：**

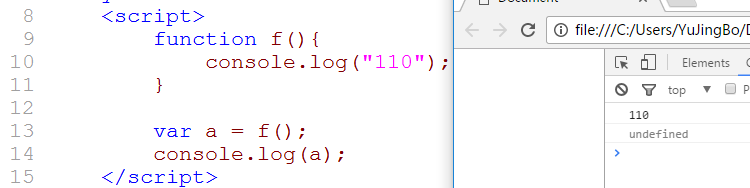


**undefined出现的地方有哪些？**

1，声明了一个变量，但是没有给这个变量赋值。



2，在函数中没有return ,默认返回undefined



**什么是编译？**

我们现在学习的大多数语言分为两类：编译性的和解释的。 如C，Java, .net C#是编译性的， 我们学习的js是解释性。我们写的C代码，计算机是不认识的，它需要转化成计算能够识别的代码，或者说可以在计算机上运行的代码，把代码从人可以识别转化成计算机可以识别的，这个过程就是编译。

我们学习的JS是不需要编译，但是它也有一个预编译的阶段。在预编译阶段做了两件事，第一件事就是扫描整个代码段，如果有语法错误，则不会进行执行阶段，而是直接报错。第二件事是如果没有语法错误，就提升，一共提升两个东西，一个变量，一个是函数。

**变量提升：**

加var , 如 var a = 110; 【提升】

不加var, 如 b = 120; 【不提升】

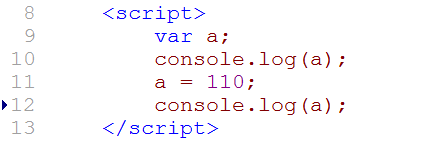
**函数提升：**

function f(){} 【提升】

var g = function(){} 【不提升】

**提升是指提升到代码段的的第一句！！！**

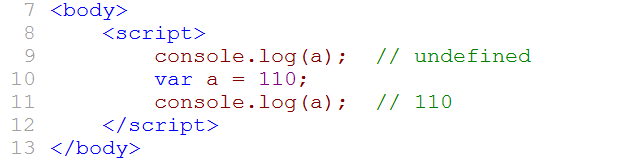
**对于上面的测试，也有一个变量提升，预编译后代码是如下这样的：**



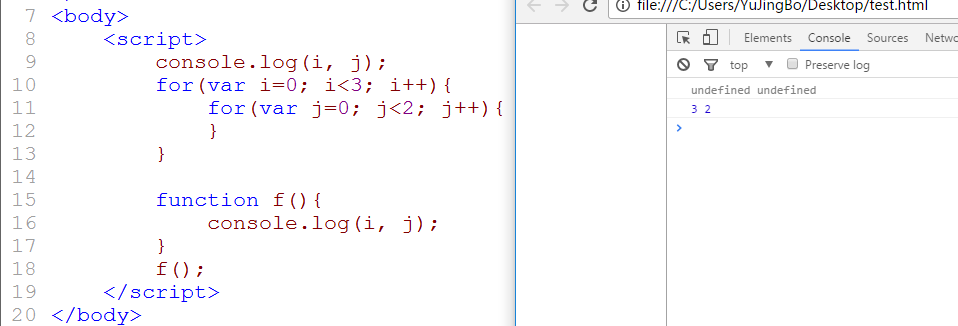
所以第一个结果是undefined, 第二个结果是110.

## 4，变量提升？

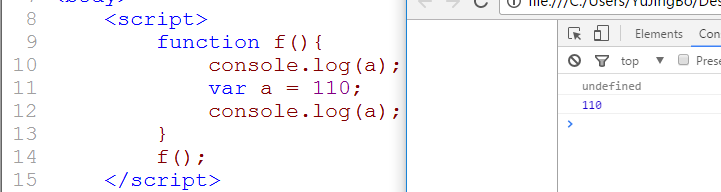
1. **普通var变量，肯定提升了**



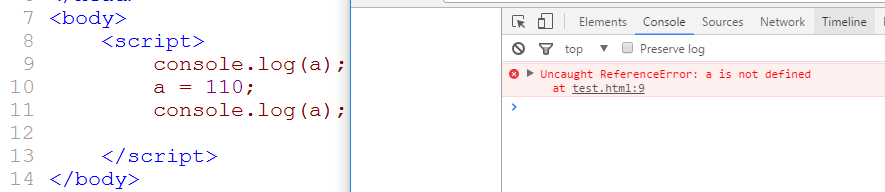
1. **循环内部的var变量， 也提升了**



1. **函数中的var变量**

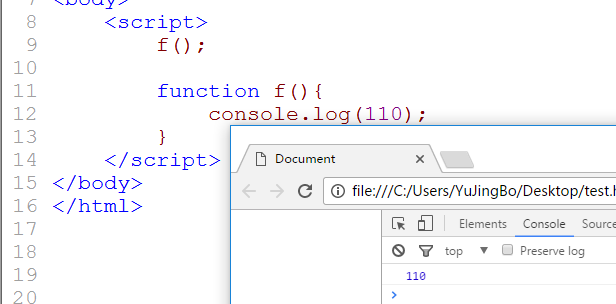


1. **遗漏声明**

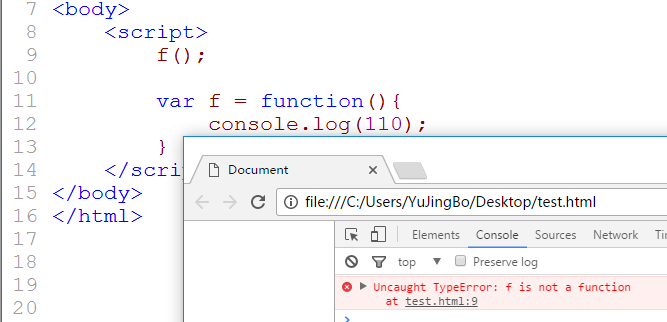


## 5，函数提升？

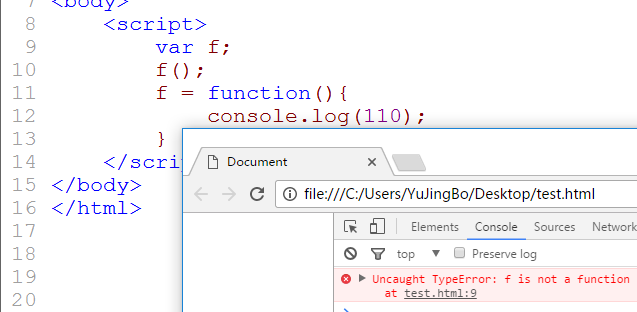
1. 函数的声明的提升



1. 函数表达式的提升

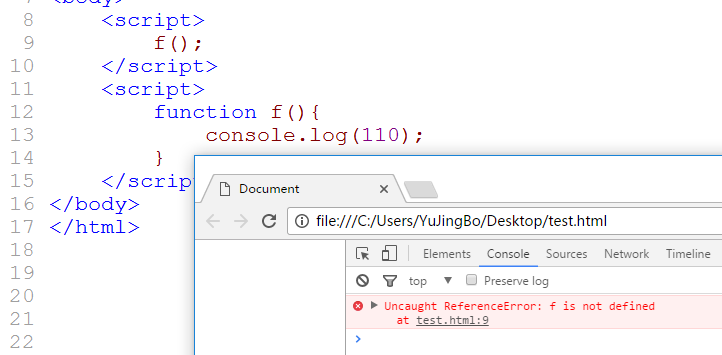


函数表达式本质上就是一个变量。预编译后的代码如下：

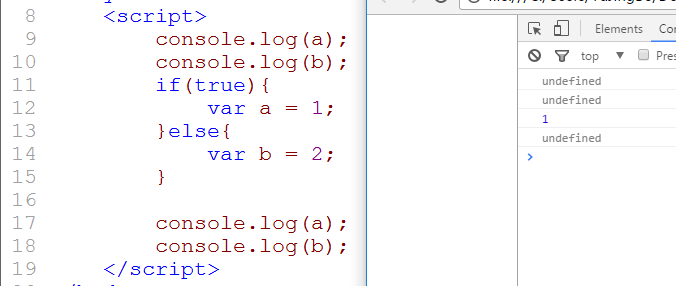


**注意的问题：**

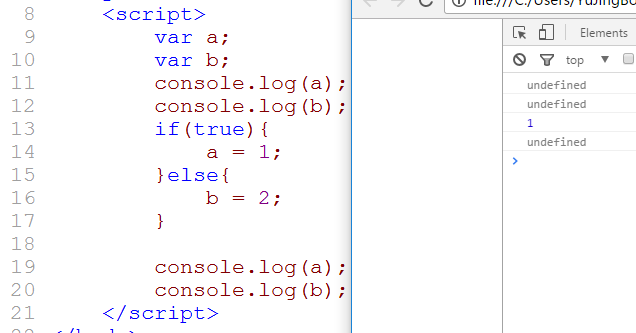
提升是发生在同一个代码段内部，它不能提升另一个代码段，否则会出现错误，如下：



**再测试一下：**

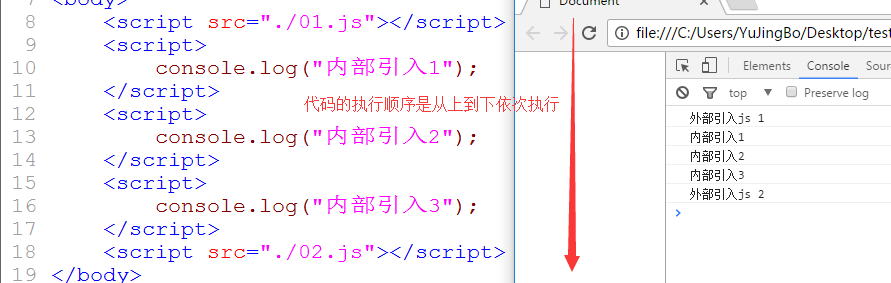


预编译后的代码如下：

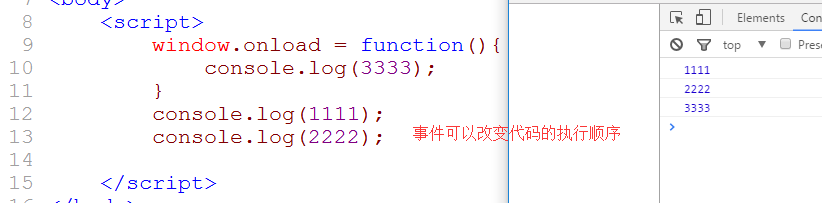


## 6，代码的执行顺序？

1，代码是从上到下顺序执行的



2，事件可以改变代码的执行顺序



**小结：**

JS是代码执行分两个阶段：

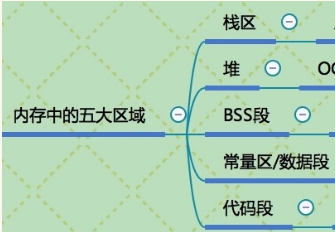
1，预编译

2，执行

# 二，数据的存储

## 0，预备内容

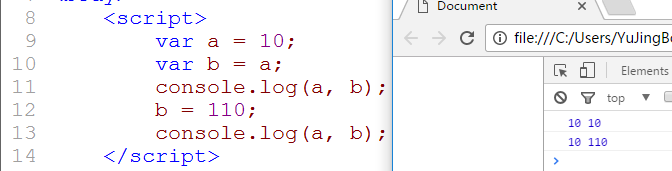
**内存五大区域：**



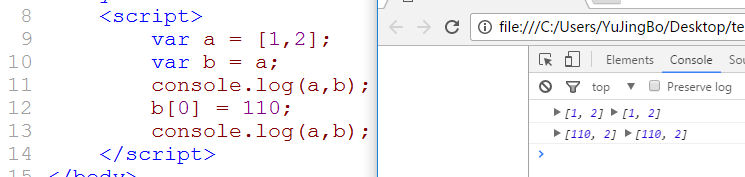
这五大区中，我们重点关注栈区和堆区。 **基本数据类型是存在栈区，对象是存在堆区。**

**测试一下：**

**测试1：**



**测试2：**



## 1，JS中都有哪些数据类型？判断数据类型的方法？

**JS中有两类数据类型：**

1，基本数据类型

5个： 数值，字符串，布尔，undefined, null

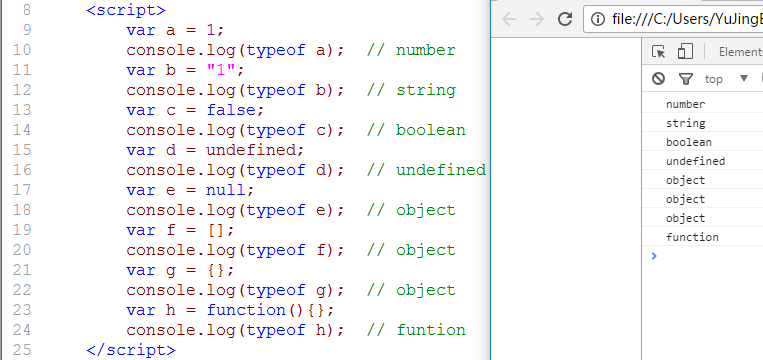
2，引用数据类型

对象：object

函数：function

数组：Array

**判断数据类型的方法：typeof**

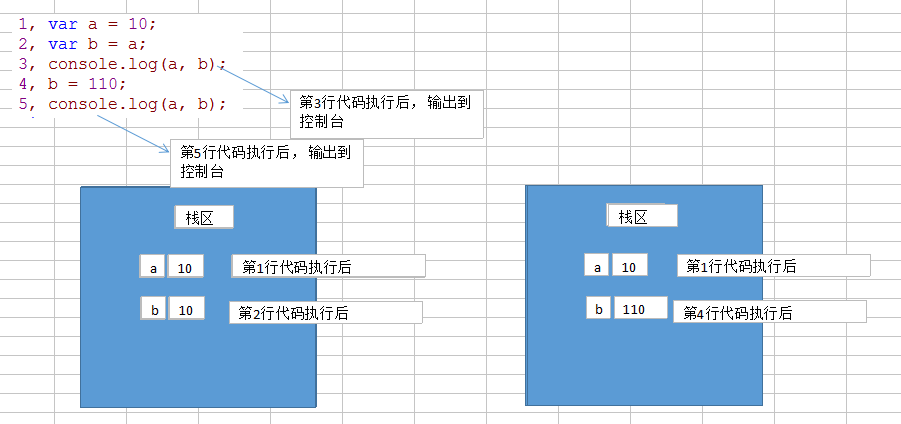


**面试题：typeof的返回值有哪些？**

number, string, boolean, undeined, object, function

## 2，基本数据类型的存储与赋值

**存储：**



**赋值：**

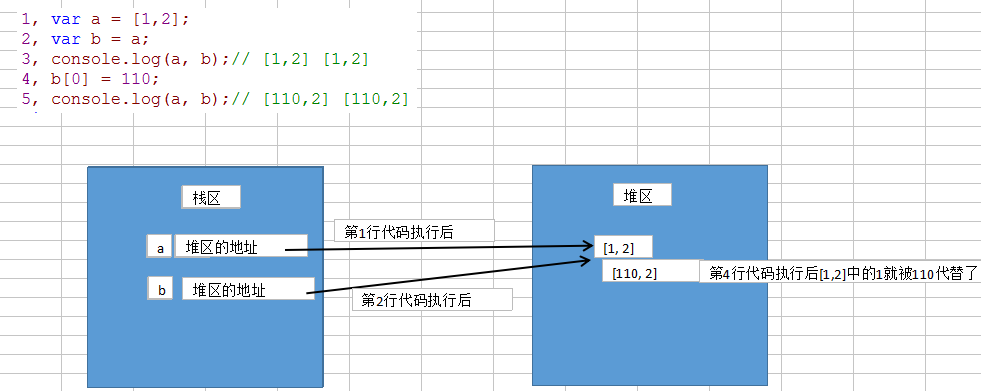
var a = 10; var b = a;

上面的代码可以这样理解：将a对应的内存中的值取出来复制了一份，给了b

注意：复制后给了b后，此时a和b对应的内存空间是彼此独立，互不影响，无论你去修改谁，都不是影响另一个。

## 3，引用数据类型的存储与赋值

**存储：**



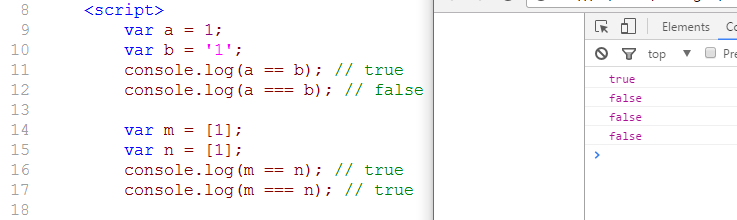
**赋值：**

var a = [1,2]; var b = a;

把a中在栈区存储的地址赋给b, 此时，a和b指向了同一个堆区中的数据。所以你对a,或b的修改，势必要影响另一个。

## 4，深入理解 == 与 ===

**再测试一下：**



**对于基本数据类型的**

a == b

先判断a,b的数据类型，如果数据类型一致，直接比较它们保存在栈区的值。如果类型不致，先进行数据类型的转换，然后在比较。

a === b先判断数据类型是否一致，不一致，就直接返回false, 如果一致，再判断值，相同 返回true, 不同返回false

**对于引用数据类型的**

a == b

没有类型比较这一步，因为它们都是同一个类型，叫引用类型，直接判断保存在栈区的值，栈区的值是指向堆区数据的地址，然后判断这两个地址是否一致。

a === b

和 a == 完全一样。

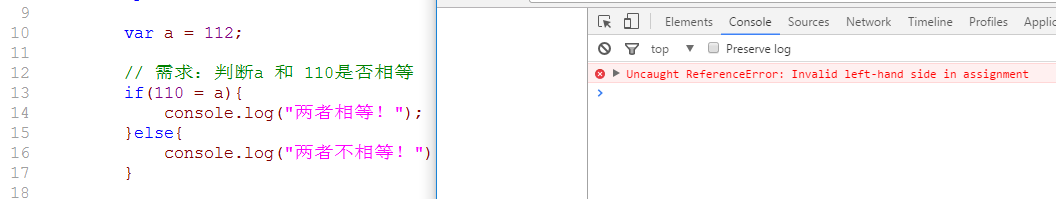
**当你想比较两个值时，最好怎么做？**

**最好习惯：**

**1，使用 ===**

**2，把字面量写在左边**

如果这样写了，容易检查出问题出在什么地方，否则，不报错，结果也是错误了，不容易找到错误。



**复习一下：**

1，什么是代码段（代码块）? 它的特点是什么？

一个script标签包含起来的代码，就是一个代码块。 独立性，共享性

2，js是以代码段为单位进行执行的。

3，预编译做两件事：一个是扫描代码段看看有没有错误。二提升。

4，变量提升：var声明的全局变量是提升的，for循环中的带var变量也是提升了，函数内部的带var变量提升了， 遗漏声明的变量是没有提升。

5，函数目前我们分两种：函数声明和函数表达式， 函数声明会提升，函数表达式不会提升。

6，js代码以代码段为单位，从上到下顺序执行，事件会改变代码的执行顺序。

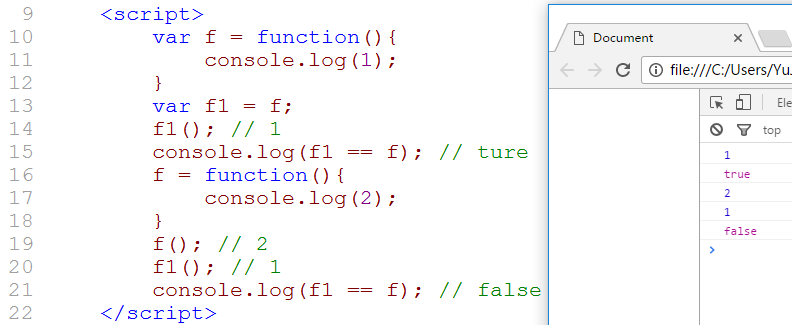
7，typeof的返回值都有哪些？ number , string, boolean, undefined, object , function

8, 基本数据类型直接将数据保存在栈区，引用数据类型的真正数据是保存在堆区的，在栈区保存着引用数据类型数据的地址。

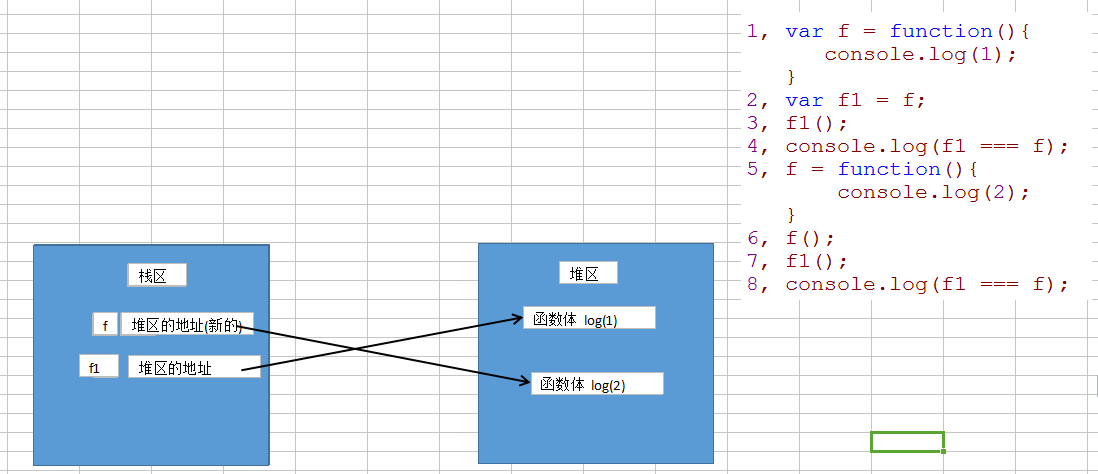
9，对于基本的数据类型的数据： a == b a===b

10，对于引用数据类型 a == b a === b

**复习测试题：**



**分析如下：**

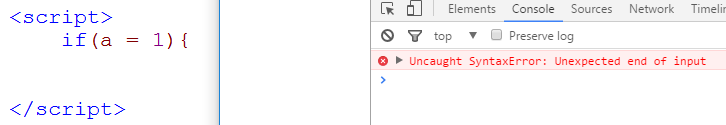


# 三，JS中常见的错误

JS中的错误分两类，代码错误（浏览器报错）， 逻辑错误（浏览器不报错）

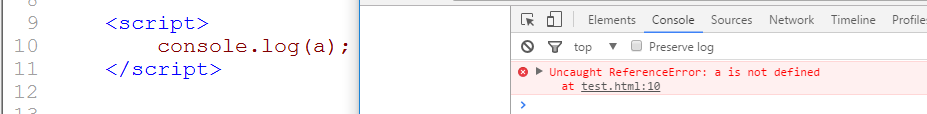
## （1）代码错误，浏览器直接报错

### 1，语法错误（SyntaxError）：



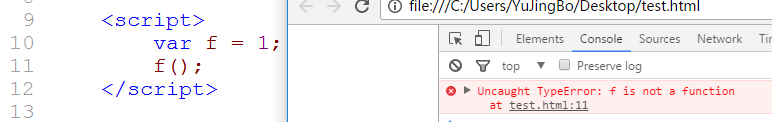
错误原因：你的代码不符合JS规则的要求。这种错误最好解决。

### 2，引用错误(ReferenceError)：



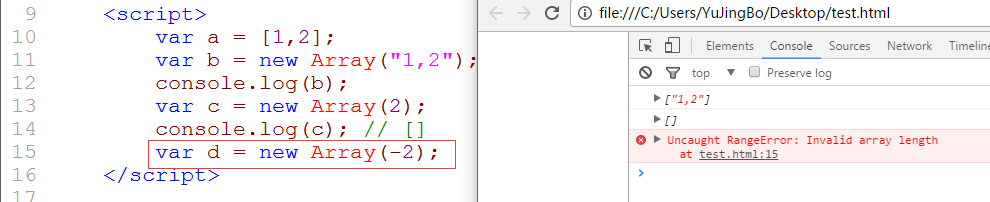
错误原因：你访问了一个前面没有定义的变量。 解决：去前面定义一次。

### 3，类型错误(TypeError)：



原因：你提供的数据，不是js想要的类型。 解决：使用正确的类型。

### 4，范围错误(RangeError)：



**总结**：

**这四种错误的相同之处：**

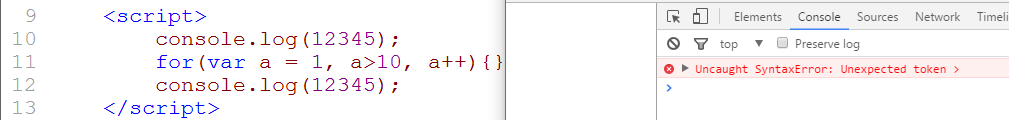
浏览器会报错，会给我们提示信息，方便我们改错

**这四种错误的不同之处：**

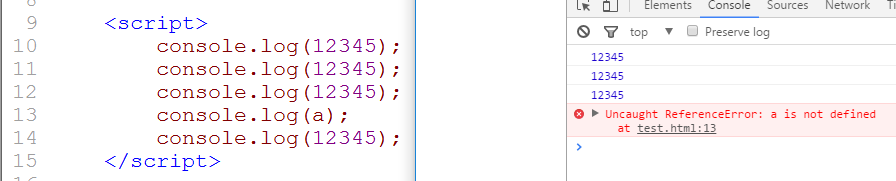
语法错误会让整个代码都不执行， 它是在预编译期间就发现了语法错误。

引用错误，类型错误，范围错误只会在出现错误的地方，以及后的代友不执行，前面的代码还是会执行的。

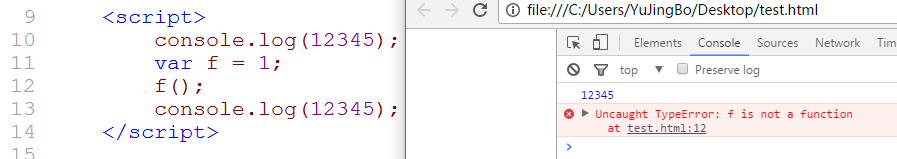
**语法错误（整个代码都不执行）：**

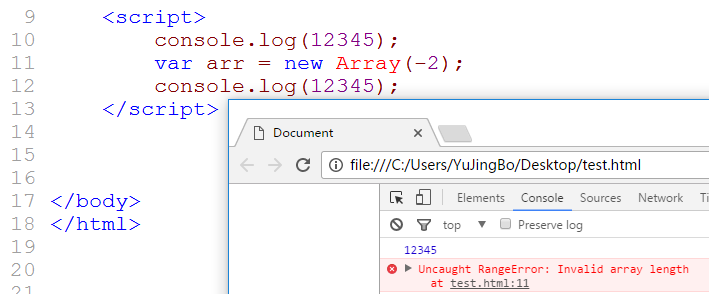


**引用错误（错误的地方以及后面的代码不执行）：**



**类型错误（错误的地方以及后面的代码不执行）：**

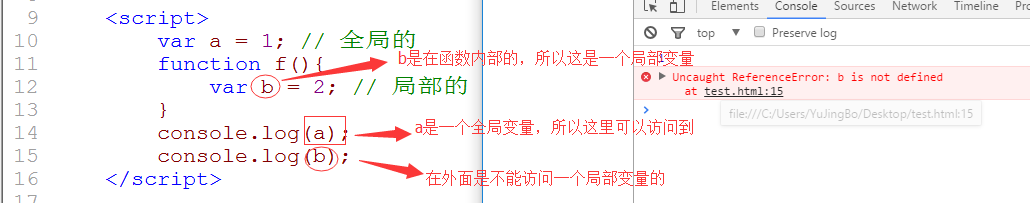


**范围错误（错误的地方以及后面的代码不执行）：**

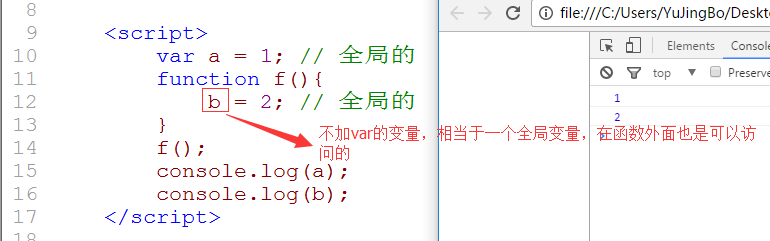
# 四，变量深入

## 1，加var与不加var的变量的区别？

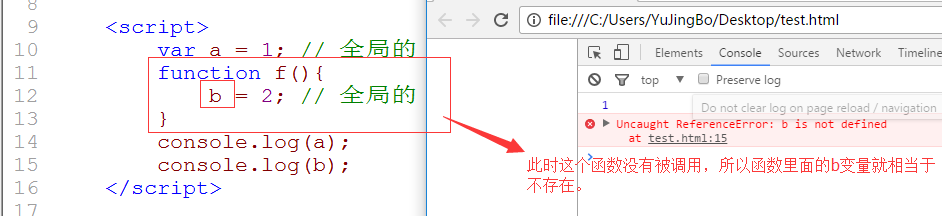
**1，加var的可以做局部变量也可以做全局变量，没有var只能做全局变量**



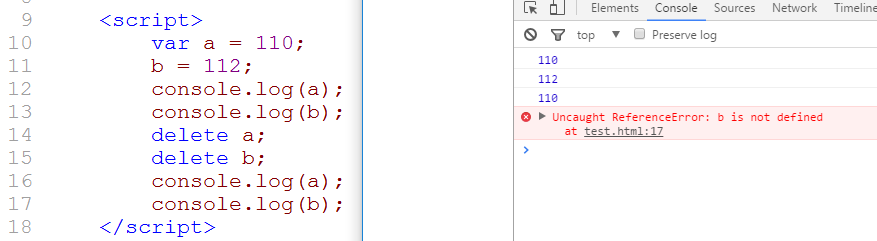
再看一个，没有var的变量是一个全局变量，在函数外面是可以访问的。



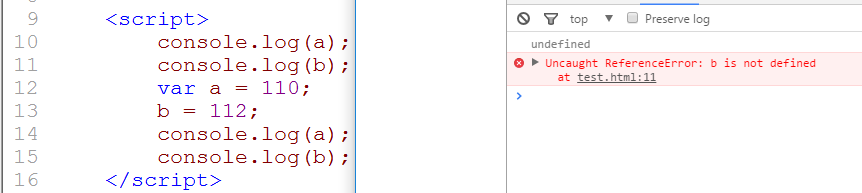
再再看一个，在函数内部中的全局变量，要想让它起作用，需要调用这个函数。



**2，是否可以被删除， 加var的不能被删除，没有var的可以被删除**



**3，加var的可以提升，没有var的不能提升。**

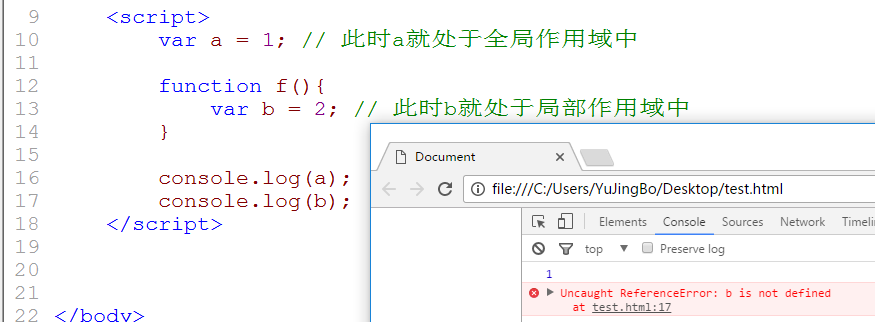


## 2，变量的作用域？

在JS中变量的作用域只有两个，一个是全局作用域， 一个是局部作用域，在JS中没有块级作用域。

### 全局作用域 （普通话）

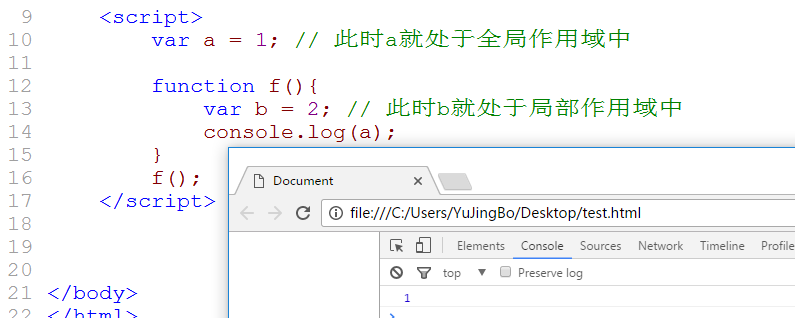
一般认为在函数的外部就是一个全局作用域



全局作用域不能去访问局部作用域

### 局部作用域（方言）

一般认为在函数的内部就是一个局部作用域

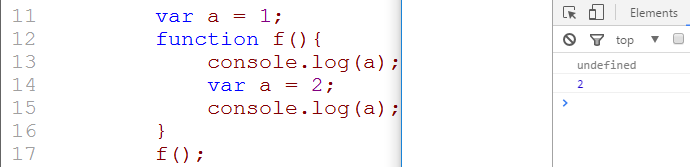


局部作用域可以去访问全局作用域

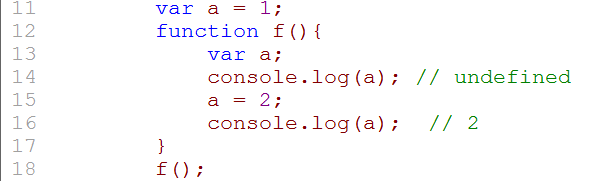
### 在JS中没有块级作用域

## 3，变量的就近使用原则

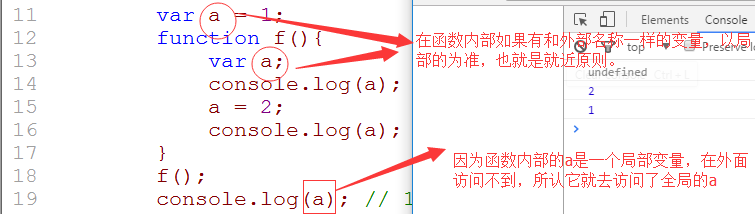
**先看一个试题：**



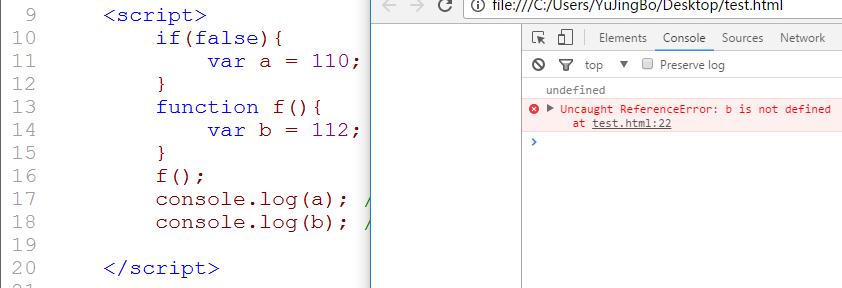
**在函数内部，带var的变量会提升，提升后的代码中这样的**：



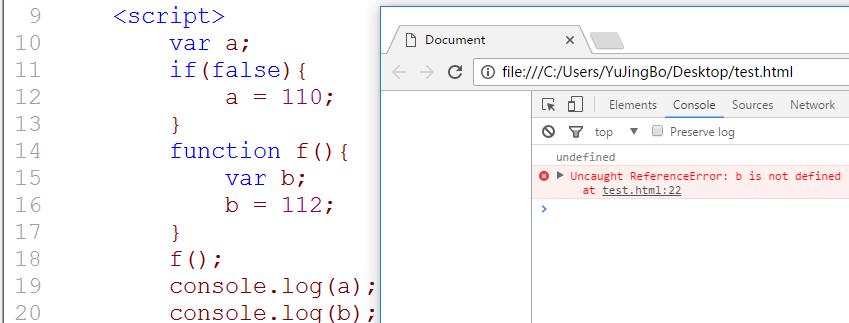
**继续这个试题：**



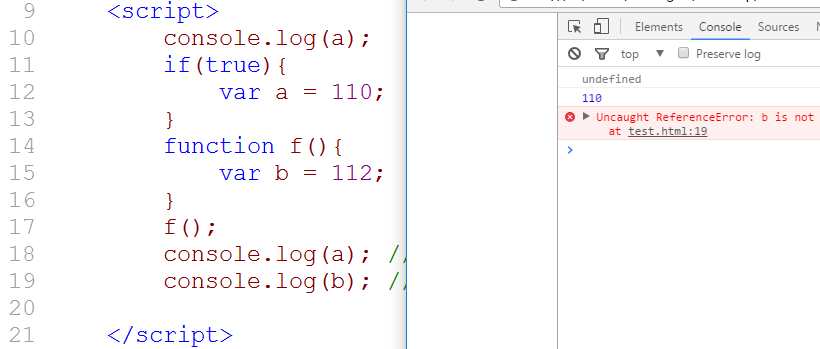
**再练习几个试题：**



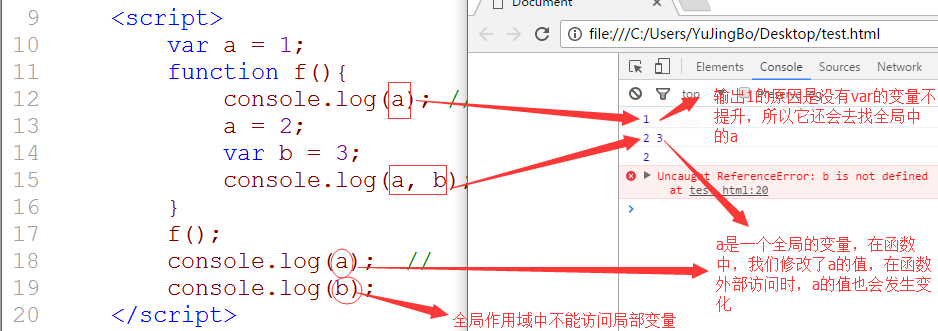
**上面练习代码的预编译之后的代码：**



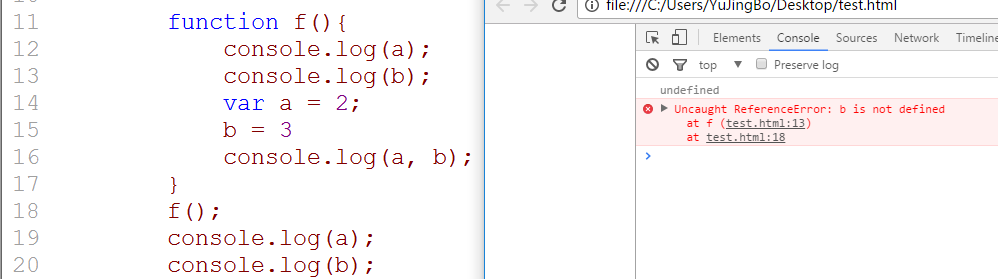
**继续上面的练习：**



**再做一个：**

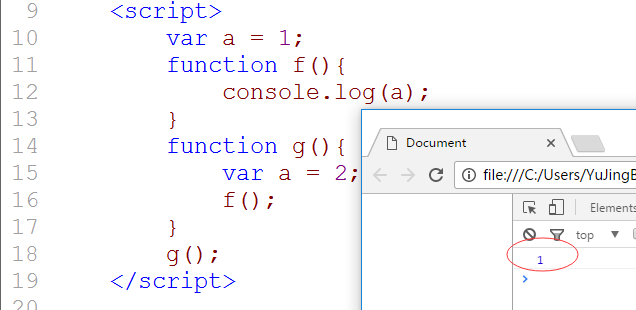


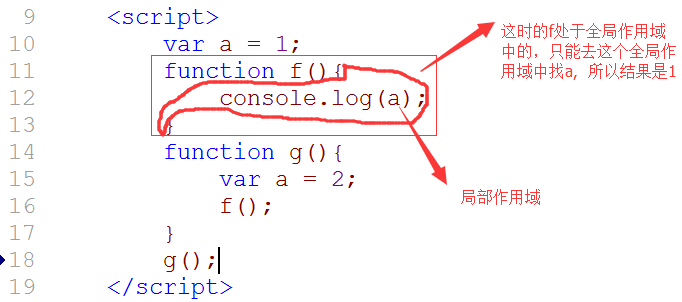
**在分析代码时，一定在注意有些错误会导致后面的代码不执行，所以写结果时，一定要分析清楚了：**



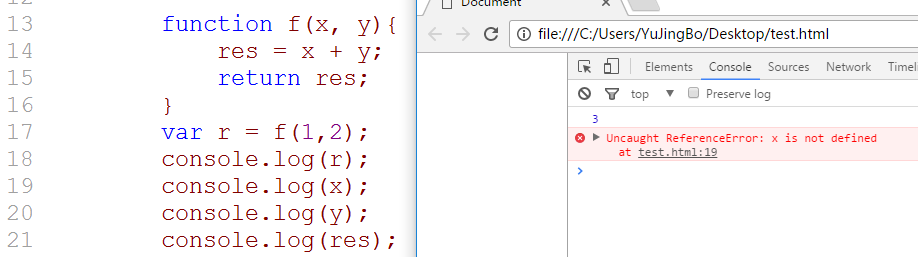
## 4，函数相互调用时，如何确定函数内部变量的值？

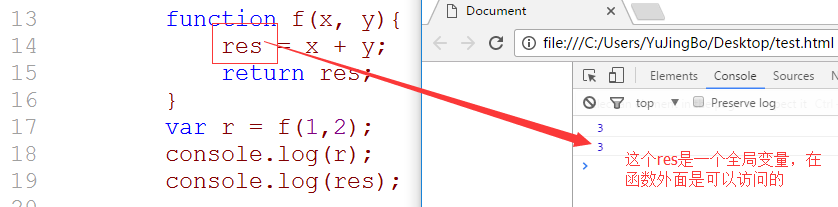
**先看一个题：**



**当你想去确定一个函数内部的变量的值时，一定要去定义这个函数的那个作用域中去找。**

**再做一个：函数的形参，你可以理解成是函数内部定义的局部变量**



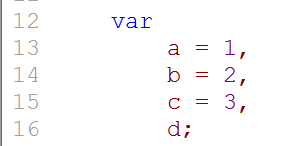


## 5，使用变量的好习惯？

1，在避免使用全局变量

解决办法是：最好不要使用没有var 的变量

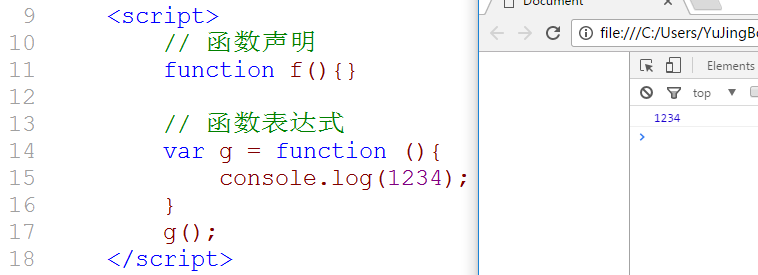
2，当你想定义多个变量，用一个var 时，将var写在代码段第一行，多个变量使用一个var.



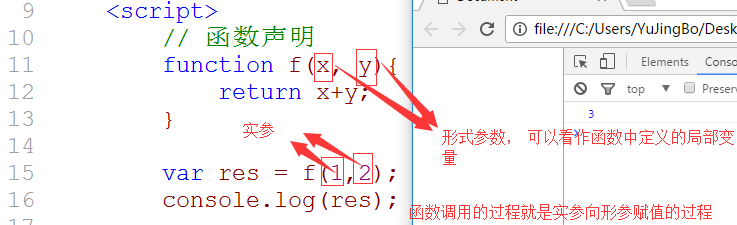
# 五，函数深入

## 1，创建函数常见的两种方式？以及函数的参数类型？

**函数声明和函数表达式：**



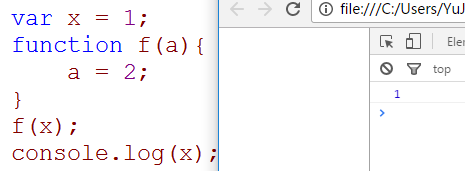
**函数参数类型：形参与实参**



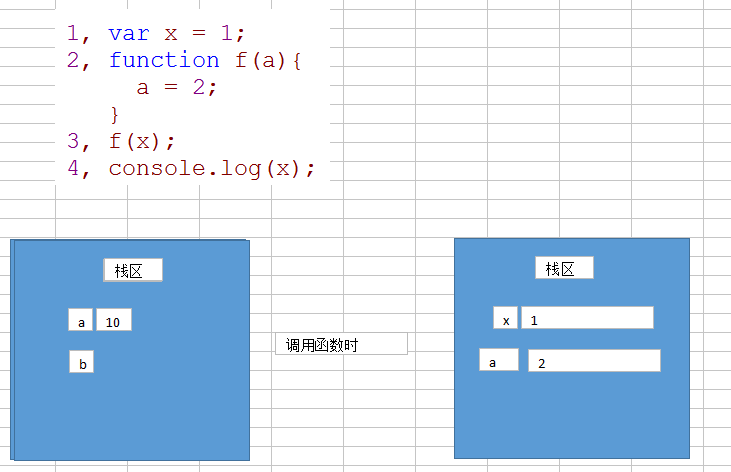
## 3，参数的传递？

### 基本数据类型的参数传递

**看一个试题：**



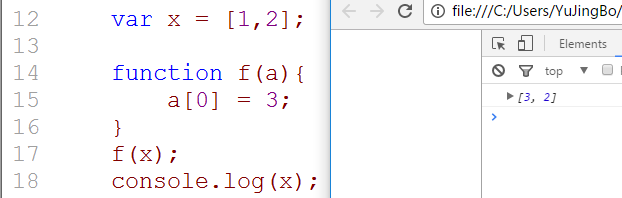
**分析如下：**



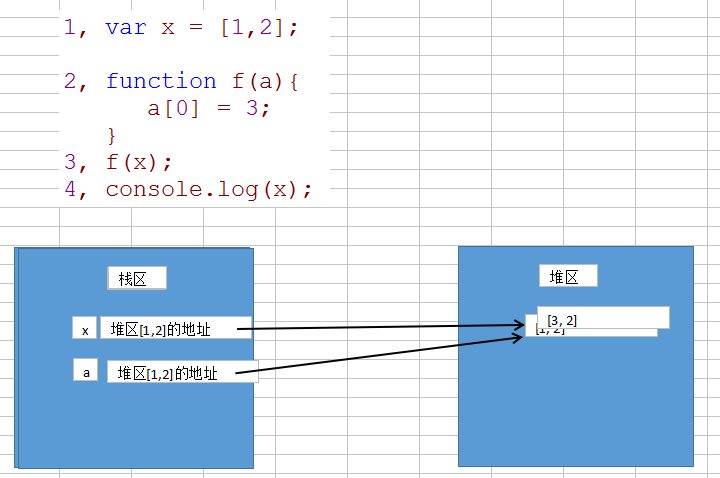
**上面的图有问题**

### 引用数据类型的参数传递

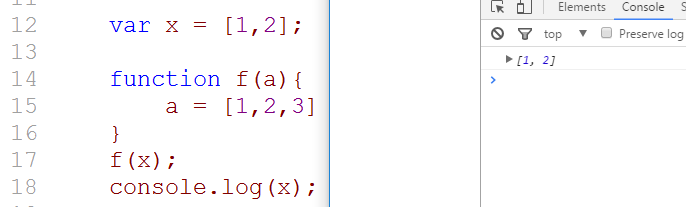
**看一个题：**



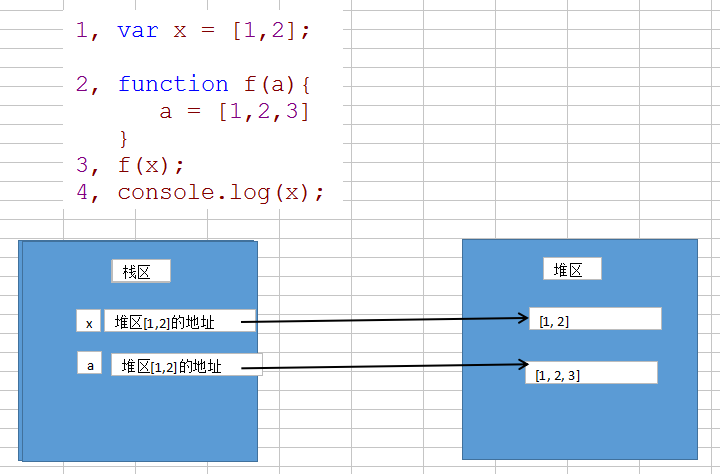
**分析如下：**



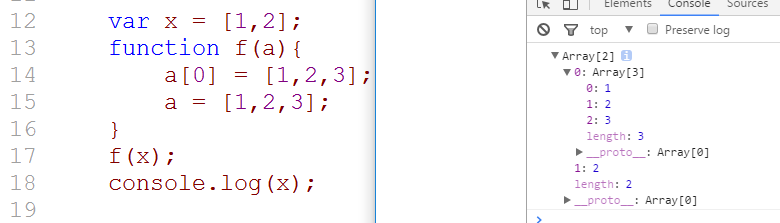
**继续上面这个试题：**



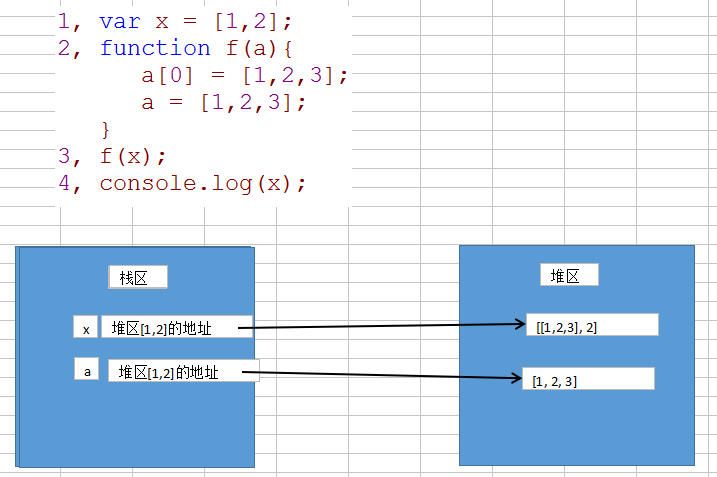
**分析一下：**



**继续分析：**



**为什么是[[1,2,3], 2]?**



**小结：**

1，函数的形参可以看作是函数体中声明的局部变量

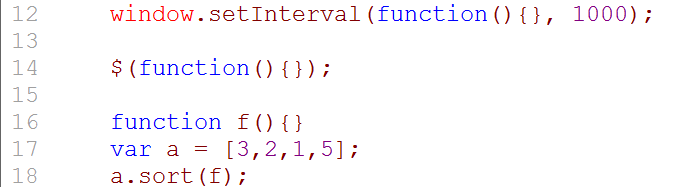
2，所有的传递都是传递的栈区中的值。

3，对于基本数据类型，是把真正的数据复制一份，传递给形参。

4，对于引用数据类型，传递的是堆区的地址。

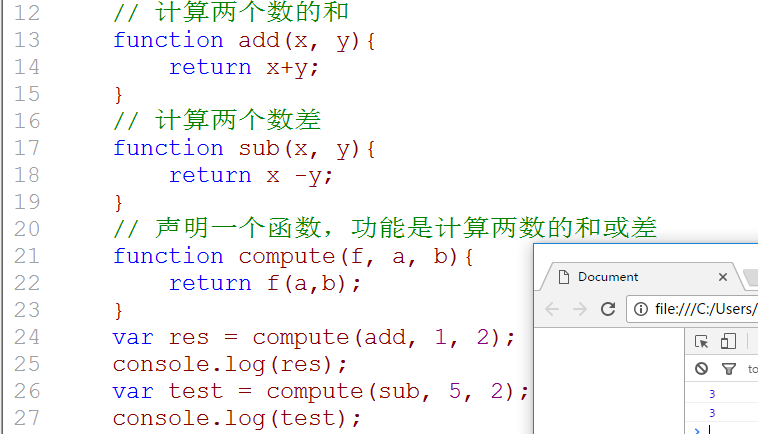
## 4，当函数的参数是另一个函数

**我们前面的学习见过很多这样的，如下：**

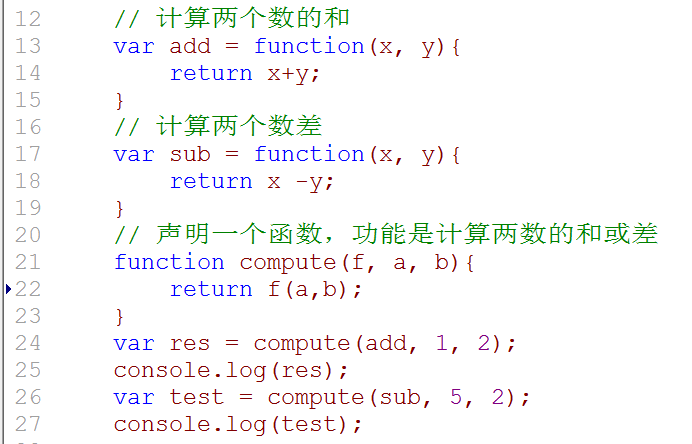


**把一个函数当作另一个函数的参数有如下几种写法：**

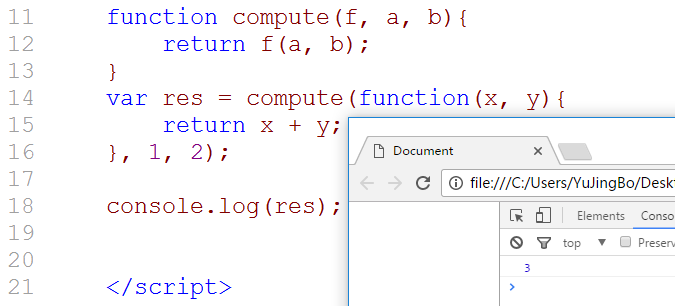
**第一种：**



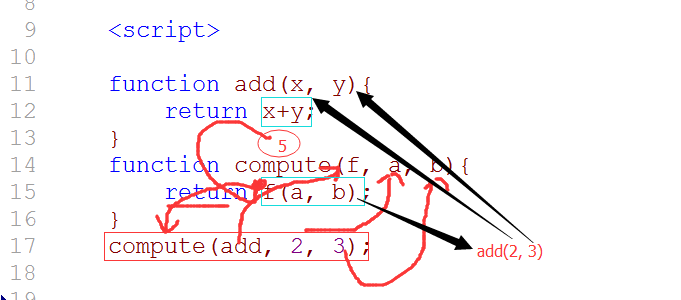
**第二种写法：**



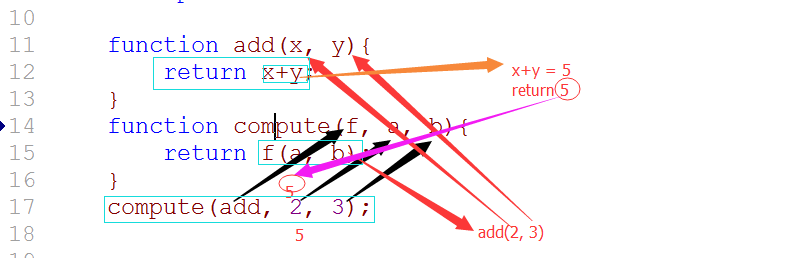
**第三种（最常见的写法）：**



**当一个函数是另一个函数的参数时，参数是怎么传递的？ 分析如下：**



**自己下去好好看一下：**



**案例练习：**

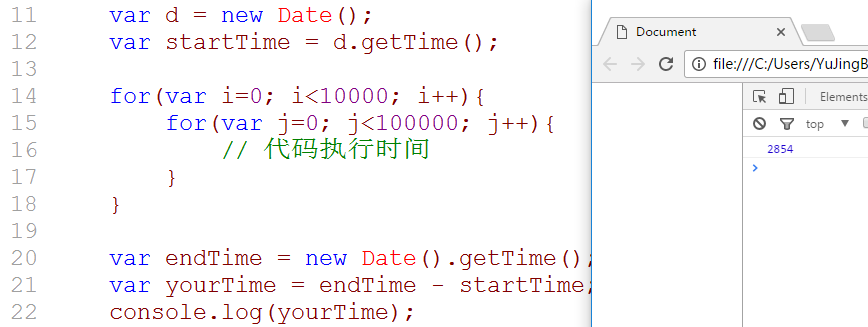
封装一个函数，函数的功能是计算另一个函数的执行时间。

**分析：**

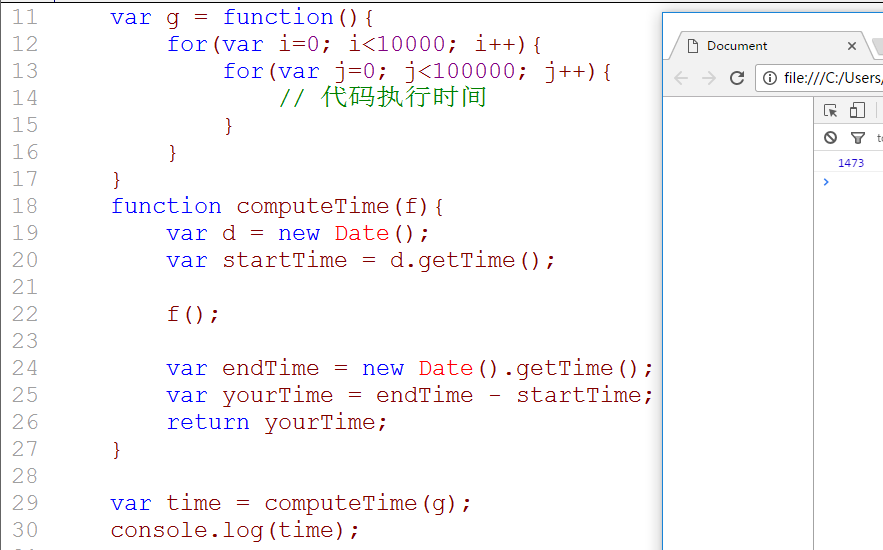
怎么去得到一个代码段的执行时间？

* 第一步，记下当前时间， startTime
* 第二步，让代码跑起来
* 第三步，记下代码跑完的时间, endTime
* 第四步，花费的时间：yourTime = endTime-startTime

计算一个for循环运行时间的代码如下：



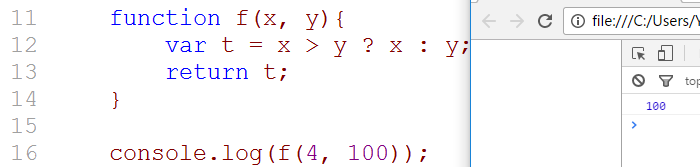
写一个函数，这个函数的功能是计算另一个函数执行的时间？

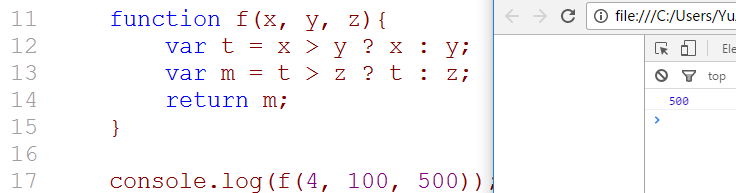


## 5，理解arguments对象

1，需求：

**计算一个一个函数中两个参数中最大值？**



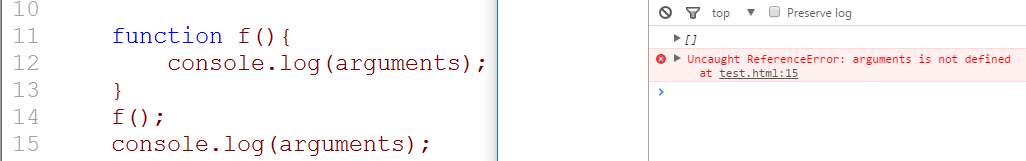
**当计算三个参数中最大值时，怎么做？**

问题：当参数个数有很多时，怎么办？

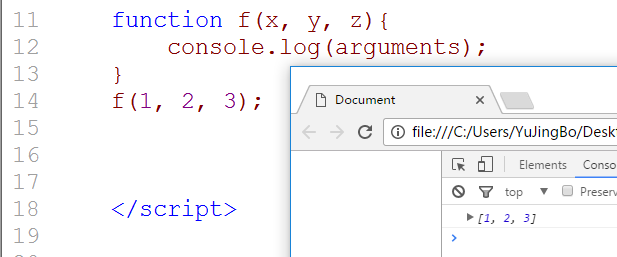
**arguments**

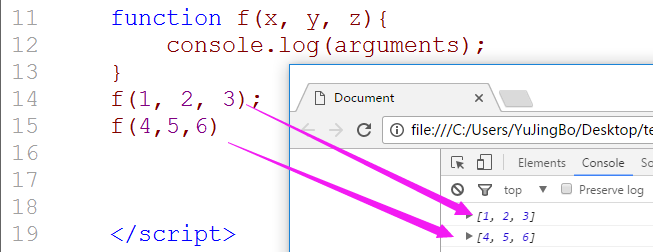
**什么是arguments?**

**是函数内部的属性，只能中函数内部使用，不能在函数外部使用**

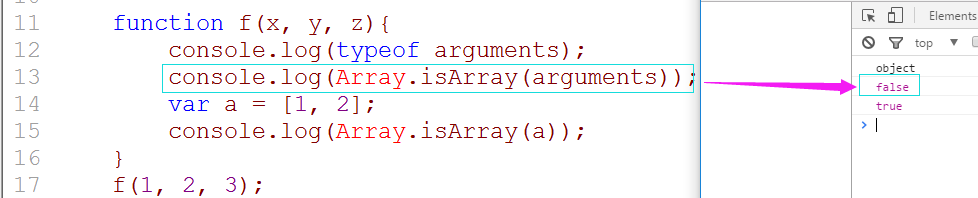


**arguments表示当前函数调用时，传递给函数的实参。换句话说，是把所有的实参收集起来，放到了arguments当中。**



**只要一调用函数，内部就会自动有一个arguments属性，这个属性保存着本次调用所传递过来的实际参数。**

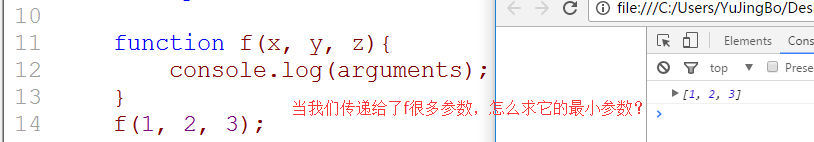
**arguments只是长的长的像数组，但是它不是一个数组。可以使用Array.isArray**



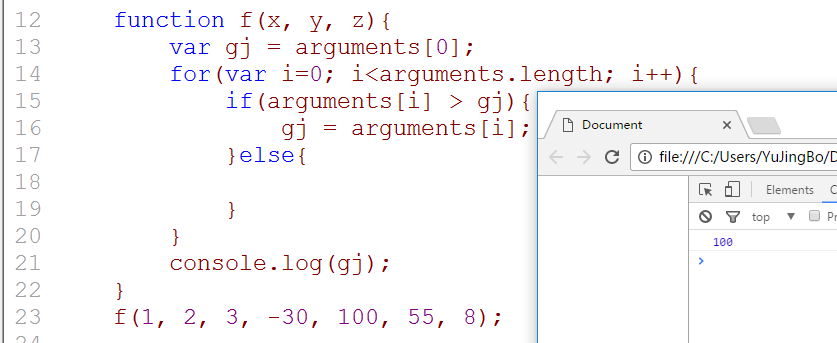
## 6，arguments对象的应用

### 求最小值，最大值

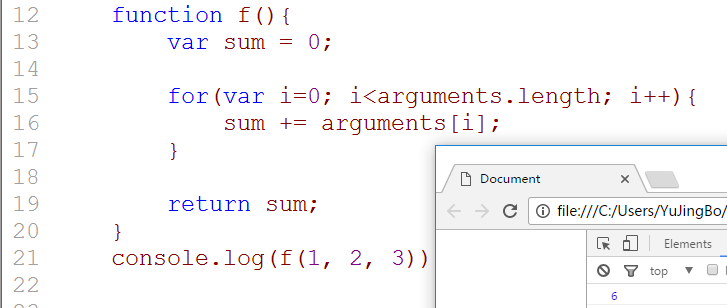
需求：



求最大值如下：最小值自已写一下：

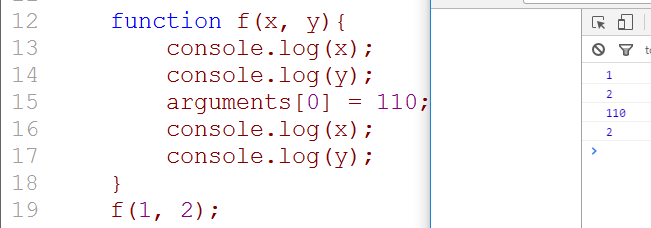


### 累加



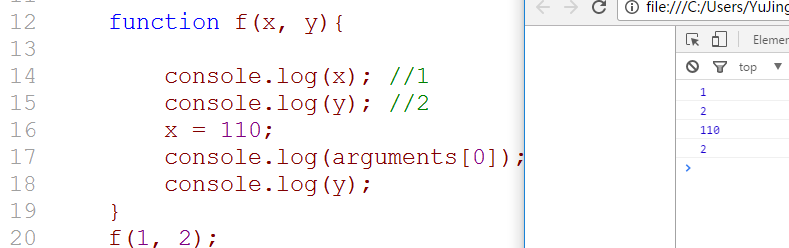
## 7，arguments与形参之间的关系

先看一个题：当你改变了argumetns中的值后，形参也改变。



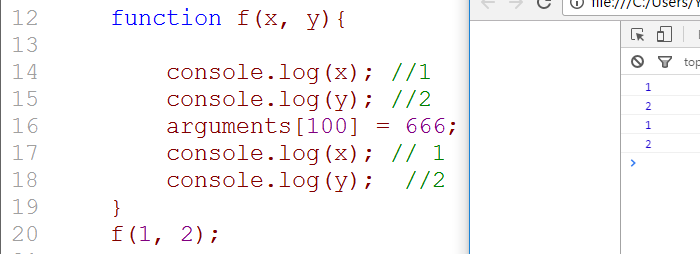
如果你改变了形参，那么arguments对应值是否改变？

结论：当你的形参改变时，那么arguments对应的值也改变，代码如下：



**综上所述：当形参列表（x, y）与arguments(arguments[0], arguments[1])之间存在一一对应关系时，一个被修改，另一个也会发生变化。**

**再看一下下面的问题：**



当形参列表与aguments之间没有对应关系时，一个修改，另一个不会变化。

## 8，如何设置函数的形参？

前面我们说到，我们可以通过实参给形参赋值，从而达到传递参数的目的。

但是我们又说了，可以使用arguments, 这个argumetns包含了实参的全部内容。

让我们感觉到，写形参好象没什么作用了，那么我们如何去设置函数的形参呢？是写，还是不写？

**最佳写法：**

1，在定义函数时，如果这个参数起到关键性的说明作用，那么你最好把它写上去。

2，当参数的个数不确定，类型不确定，需要在程序中动态获得时，可以不写参数。

**复习:**

**1,深入变量**

a，加var的变量与不加var的变量区别：

* 加var可以作全局变量也可以作局部变量，不加var的只能作全局变量。
* 加var不能被删除，不加var的可以被删除
* 加var的能提升，不加var的不能提升

b，变量的作用域

* 全局作用域，最外层的函数外面一般是全局作用域
* 局部作用域，函数内部一般是局部作用域
* JS中没有块级作用域，在其它编程语言，{}可以划分块级作用域

c，如何确定函数内部变量的值，如果当前函数内部没有定义局部变量，就去定义这个函数的作用域中去找那个变量。

**2，函数深入**

a，创建函数有两种方式：函数声明和函数表达式

b，函数参数的类型有两种：形参与实参

c，函数调用分两个过程：一个是实参向形参赋值的过程，一个是执行函数体的过程

d，函数的参数传送分两种：一个基本数据类型的参数传递，一个是引用数据类型的参数传递，它们传递的都是栈区中的值。

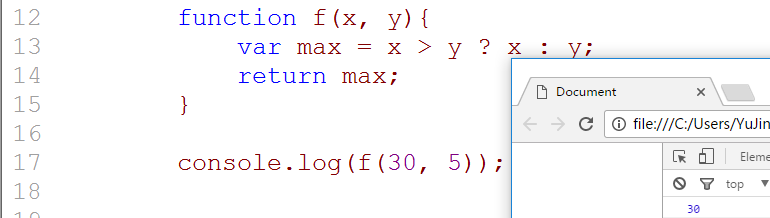
e，一个函数作为另一个函数的参数

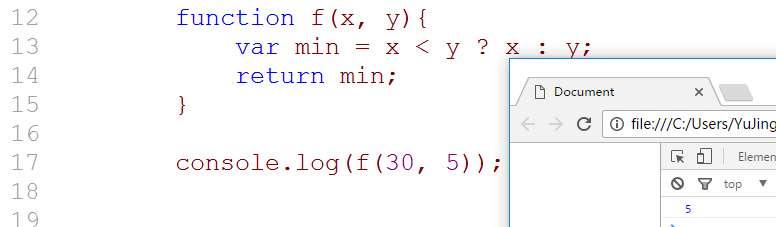
f，arguments对象：当函数调用时，会在函数内部产生一个arguments对象，这个对象收集了所有的实参，我们可以直接使用arguments[下标]来使用实参。

## 9，函数返回多个值

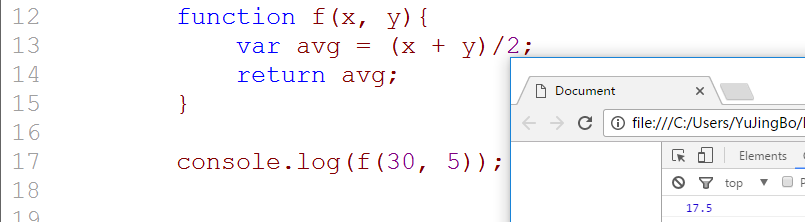
**提一个需求：**

**1，求最大值：**



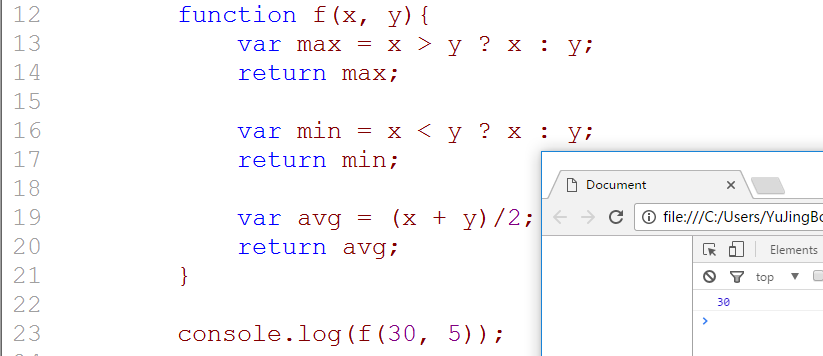
**2，求最小值：**

**3，求平均值：**

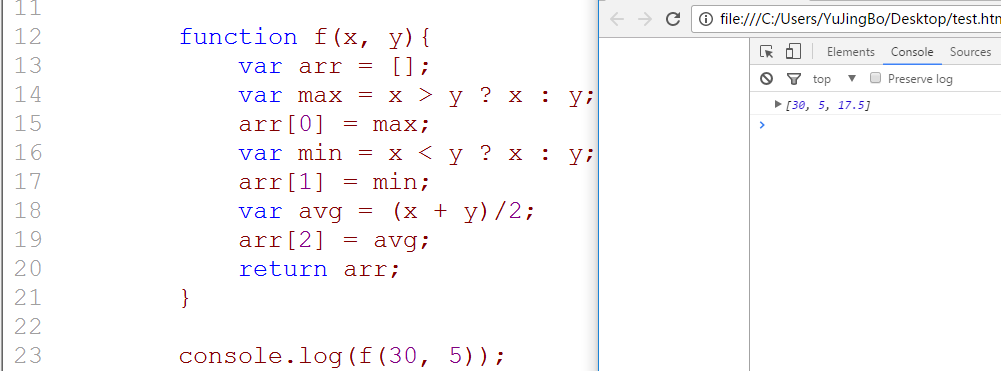


**我们能否去定义一个函数，让这一个函数去返回最大值，最小值，平均值：**

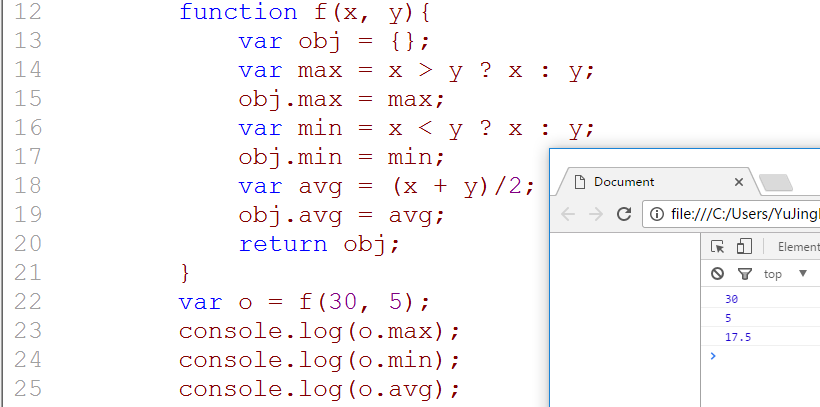
当我们按下面的写法是不可以的，因为return是跳转语句，return下面的代码肯定就不执行，所以不能实现我们的功能：



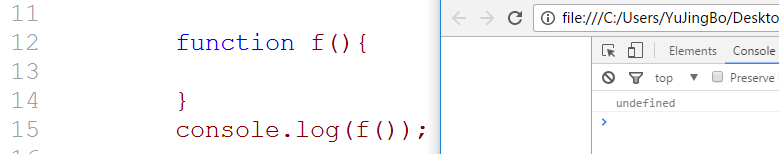
**我们可以返回一个数组，在数组中保存着最大值，最小值，平均值：**



使用对象，我们的把一个对象返回：



**如果一个函数没有写return，默认它返回undefined:**



**总结：**

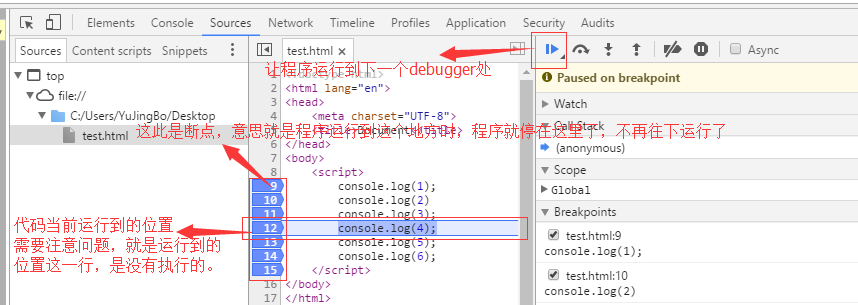
函数是不能返回多个值，只能返回一值（这个值也可以是数组或对象），如果你没有给它返回值，默认返回undefinted。如果我们想当我们调用一个函数时，让它返回多个数据，那么我们可以在函数内部去声明一个数组或对象，把多个值保存到数组或对象中，最后把数据或对象返回。

# 六，执行上下文

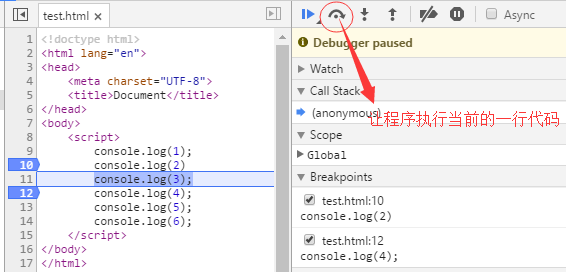
## 1，debugger的使用

我们前面说过，JS中的错误分两种：一种是代码错误，一种是逻辑错误，代码错误，浏览器去提示，我们可以根据提示去解决错误，那么逻辑错误，浏览器不提示，如何解决错误？ 就使用debugger工具！

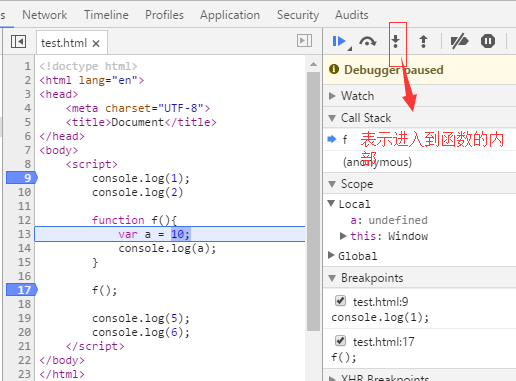
第一个小按钮说明：



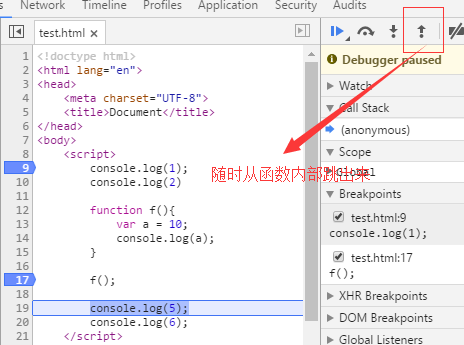
第二个小按钮说明：



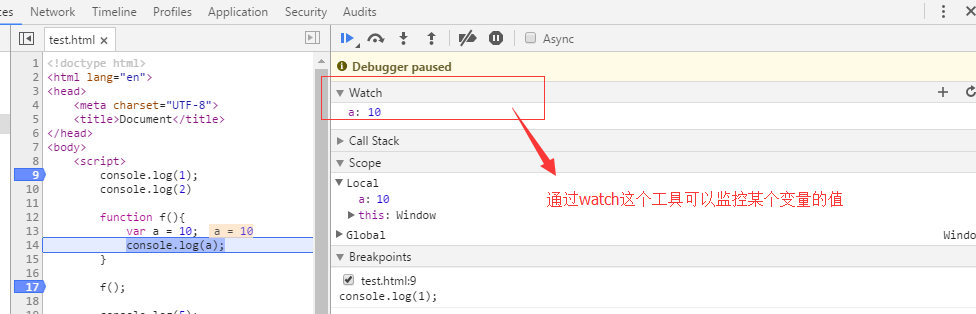
第三个按钮说明：



第四个小按钮：



watch小工具说明：



## 2，什么是执行上下文

**1，函数代码与全局代码**

一个函数我们就叫它为函数代码，当我们打开我们页面后，一上来就执行的代码，我们叫它为全局代码。全局代码只有一份，局部代码可能有很多，因为函数可能有很多。

**2，全局执行上下文，函数的执行上下文**

当执行一个函数代码时，就会产生一个函数的执行上下文，执行全局代码时，就会产生一个全局的执行上下文。

全局的执行上下文只能有一个，函数的（局部的）执行上下文可能有多个。

**3，每执行一个函数，就产生一个函数执行上下文，执行100函数，就产生100个执行上下文件。**

**4，同一个函数，如果你执行了一次，那么就产生一个执行上下文，如果你调用了100次，那么就产生100个执行上下文。**

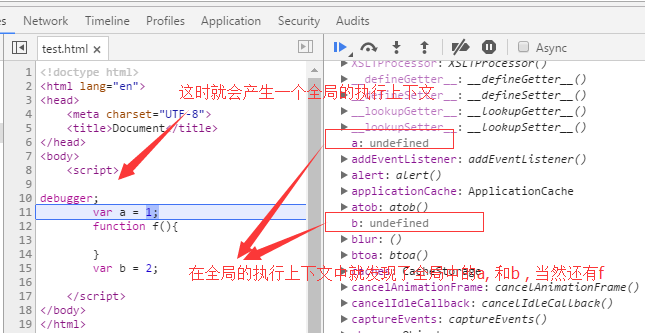
**5，执行上下文的作用：**

函数在执行的过程中，需要的一切数据都由执行上下文来提供。数据包括：变量与函数。

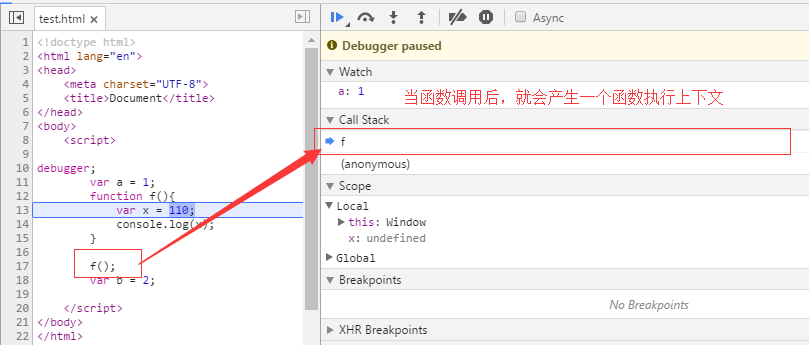
**6，程序刚开始会进入到全局的执行上下文。**

## 3，全局的执行上下文

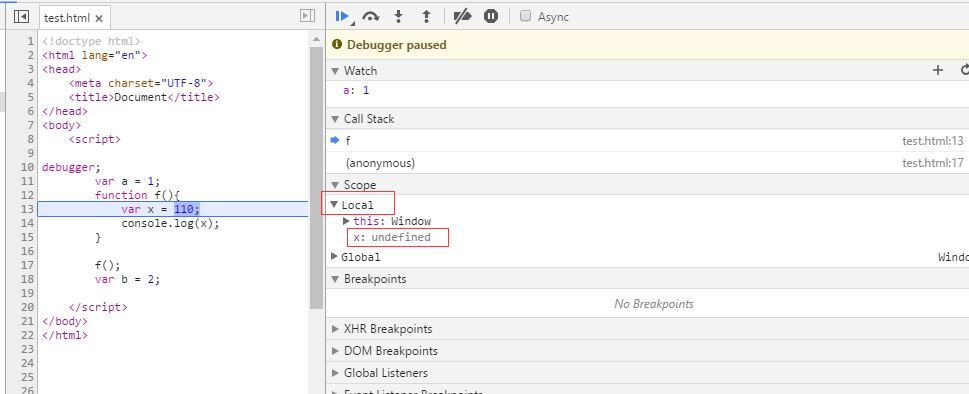
理解一个window和globle。 我们就可以把window看作是一个全局执行上下文：



## 4，局部的执行上下文



函数执行上下文提供了函数内部的数据：



## 5，执行上下文件栈

**栈：**

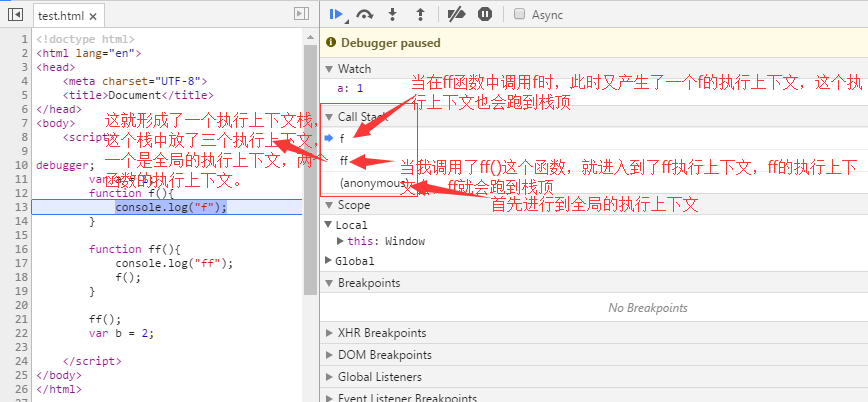
在计算机中，栈是一种数据结构，这种数据结构描述了操作数据的一方式。

先进后出

**队列：**

先进先出

**执行上下文，如下图：**



**分析 ：**

当全局代码一执行，就会产生一个全局的执行上下文(Anonymous),此时这个全局的执行上下文处于栈最下最。

当调用ff()函数时，就会产生一个ff的函数执行上下文，这个执行上下文会放到全局执行上下文的上面，也就是说它处于栈顶位置。

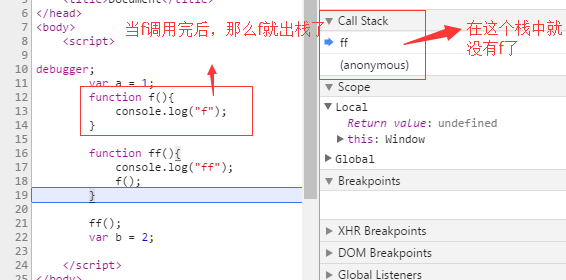
当在ff函数中，调用f()函数时，也会会产生一个f的函数执行上下文，这个执行上下文会放到ff执行上下文的上面,那么此时，f就牌栈顶位置。

**出栈和入栈：**

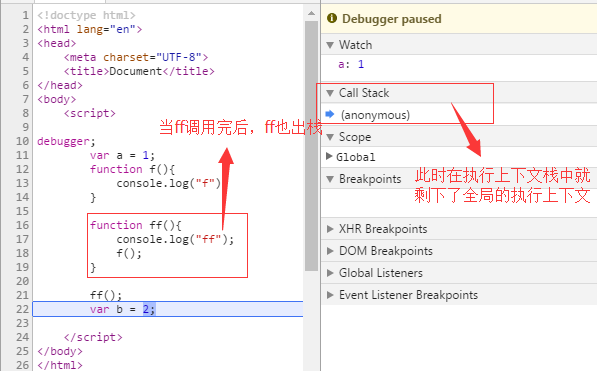
前面我们所演示的就是入栈的操作，就是全局的先入栈，然后是ff入栈，再然后是f入栈。何时出栈尼？

根据上面的代码继续分析：

**f先出栈：**



**ff出栈：一旦执行上下文出栈了，就意味着这个函数需要的数据就被回收。**



## 6，执行上下文的作用

**作用：提供代码运行时所需要要的各个数据的值。**

**函数的执行上下文包括了两部分内容：**

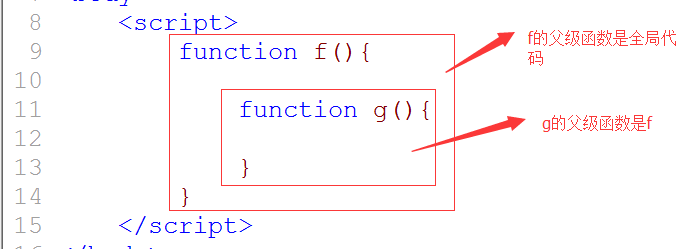
* 本函数内部定义的变量，内部定义的函数，arguments
* 本函数的**父级函数**的执行上下文。

**全局的执行上下文由一部分组成：**

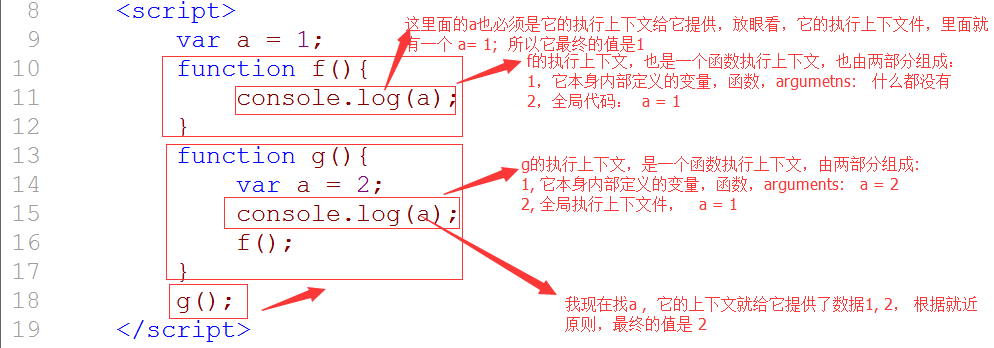
* 全局代码中定义的变量和函数

**什么是父级函数？**

在定义一个函数时，直接把这个函数包起来的那个函数就是父级函数，如果这个函数外面没有函数，那么它的父级函数就是全局代码**。**



**思考前面的见到一个问题：在函数互相调用过程中，如何去确定函数中变量的值？**



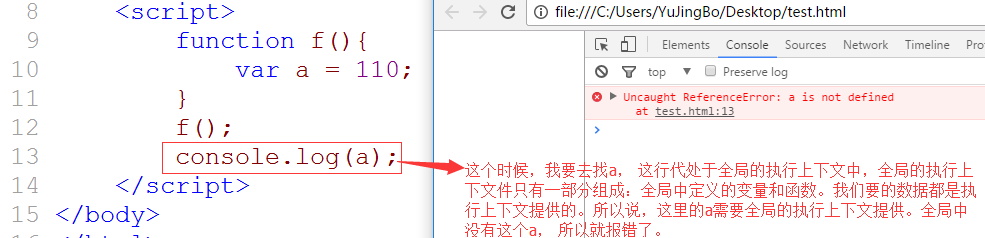
**总结：**

函数执行上下文中保存的数据有两部分组成：1，是自已函数内部定义的变量，函数，arguments， 2，它的父级函数的执行上下文。

全局执行上下文件中保存的数据只有一部分：在全局中定义的变量和函数。

**先在自己的内部去找，如果找到了，就用自己的，如果找不到，就去它的父级函数的执行上下文中去找，如果找到就用它父级的，如果找到到，就报错。**

## 7，函数内的变量在函数外无法访问的真正原因

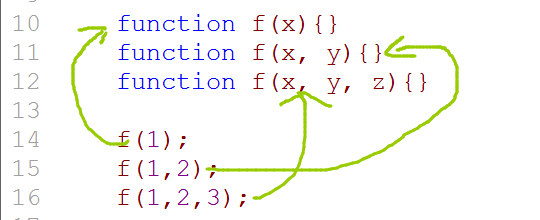


# 七，函数的嵌套定义

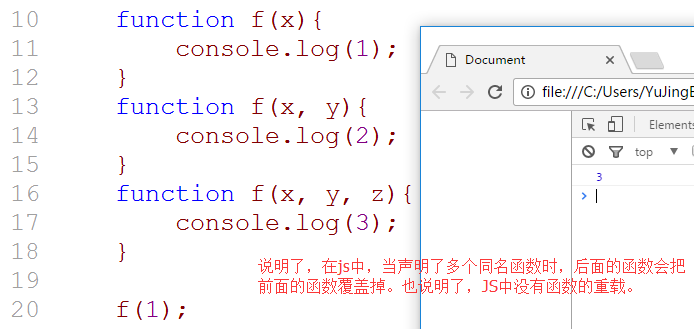
## 1，JS中没有函数重载

**什么函数重载？**

有多个函数，函数的名字相同，参数列表不同。



**JS中没有函数的重载**

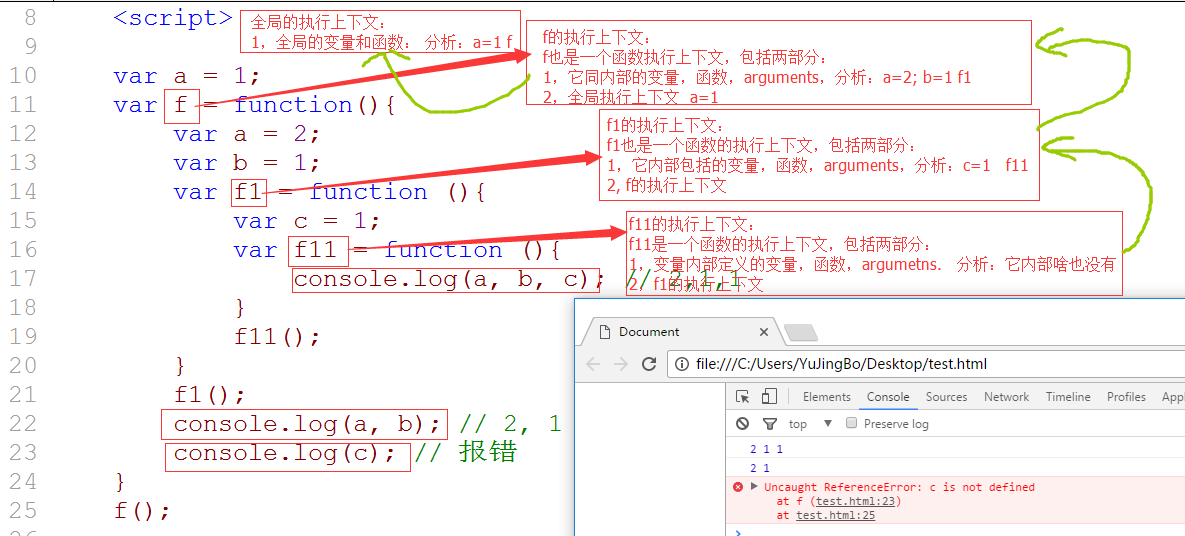


## 2，作用域链

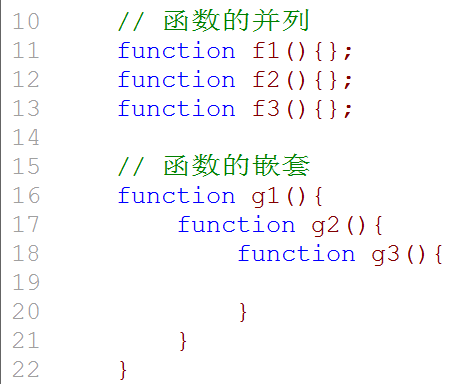
在JS中，作用域分两种：全局作用域和局部的作用域。

它们的特点是：全局的作用域不能去访问访问的作用域，局部的作用域可以访问全局的作用域。

由于函数内部又定义了函数，所以它势必要形成一个上下文从里到外的引用关系，那么这个引用关系也可以叫做作用域链，如下图所示：

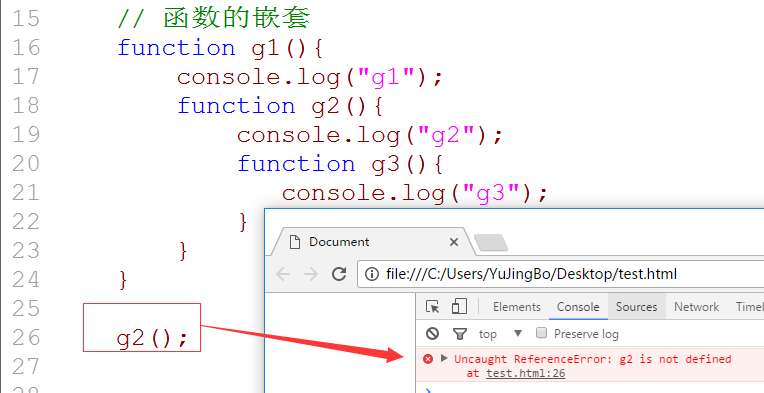


## 3，为什么我们要把一个函数定义在另一个函数内部？

一个函数是为了完成某些功能，功能肯定有大的功能，也有小的功能，对于小的功能，可能一个函数就搞定了，对于大的功能，有可能一个函数搞不定。一个函数搞不定，那么我们就需要去定义多个函数，定义多个函数有两个方式：函数的并列，函数的嵌套。如下图：

**函数的嵌套的好处：**

1，封闭性，定义在g1内部的函数，在g1外面是不能访问的。



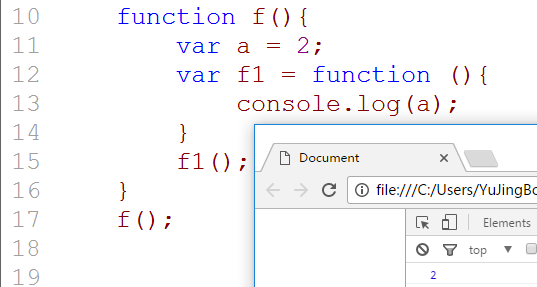
2，如果把g2,g3写在外面，那么有时候g1可能不能使用g2, g3。

**函数并列坏处：**

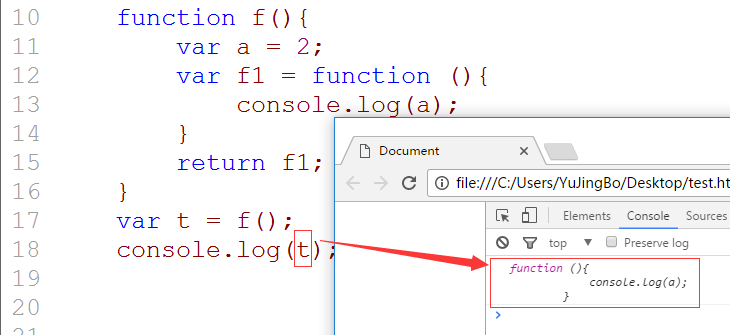
可以暴露函数的实现细节，如果写在了外面，可能无法直接使用子函数。

## 4，把函数作为返回值

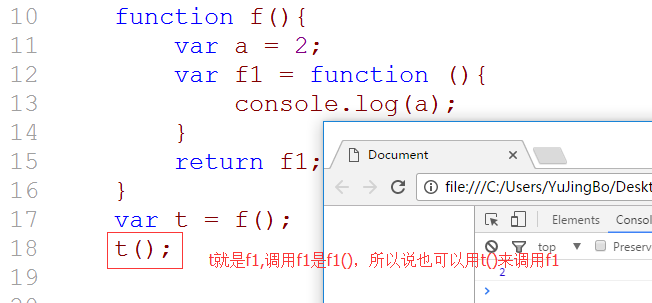
分析以前的代码如下：



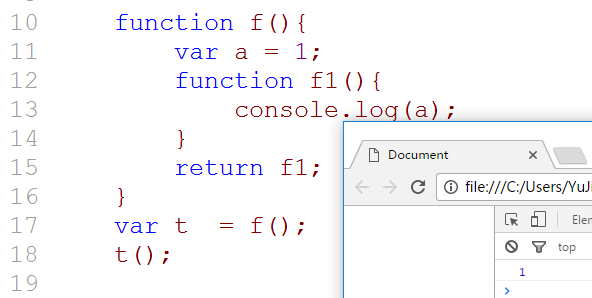
我们可以把一个函数给返回了：



当把一个函数给返回了，我们可以通过接收它的那个变量来调用这个函数：



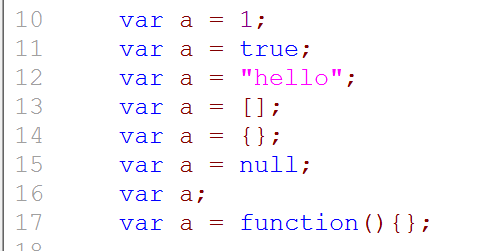
**上面我们分析的是函数表达式，接着分析一下函数声明：**



**总结：**

1，函数也是一种数据类型，函数也是一个对象，对象是引用类型。

2，var a = 它的后面可以存储任意数据类型。



3，function f(){};函数声明 var f = function(){}函数表达式。对于函数声明，我们也可以这样认为：function f(){} 定义了一个变量名字叫f, 值是一个函数体。

4，函数调用： 函数名（）;

5，在函数中，当你返回了一个函数时，那么返回的是函数的名字，说白了，就是返回了一个栈区的地址，那么这个地址是指向堆区的具体函数。

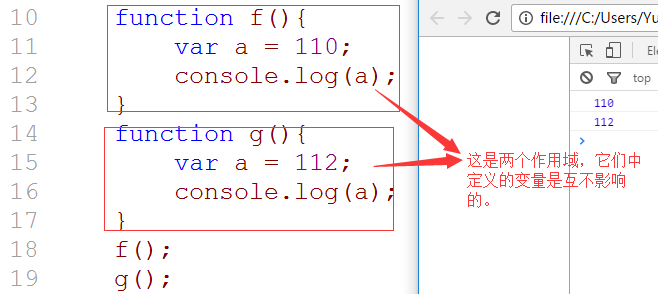
6，对于函数的嵌套，我们主是把嵌套的函数给返回了，返回完，主要是在外面使用它。

# 八，立即执行的函数表达式（IIFE）

## 1，当多人协作开发时，遇到的问题？

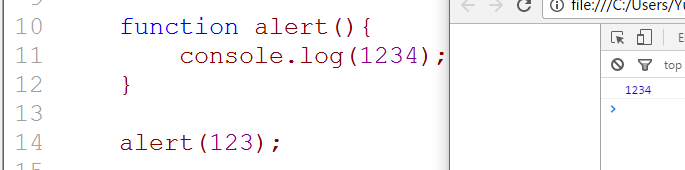
在实际开发中，我们会与其它人一起写代码，那么就会有一个问题：你的代码中用到变量或函数名，可能与其它人的代码冲突。如何解决冲突？

第一种方案：把你写的代码放到一个函数内，然后，去调用这个函数。函数可以划分作用域。也就就，你定义的这个函数中的变量，函数名，函数外面是不能访问的。所以就不会覆盖另人代码中的变量，专业的说是不会污染全局变量。



我们知道，js中有一个alert(), 我们能不能写一个alert(), 这两个alert()函数互不影响：

当我们写一个alert()函数时，那么系统的alert()函数不能用了：



**怎么解决：将我们的自己写的alert函数放到另一个函数中**



问题又来了：如何最外面的函数f也冲突了，怎么办？

**解决办法：立即执行函数表达式。**

## 2，立即执行的函数表达式三种实现方式

函数表达式：

var f = function (){}

前面我们接触过下面的写法：

btn.onclick = function (){}

setIntervar(function(){}, 1000);

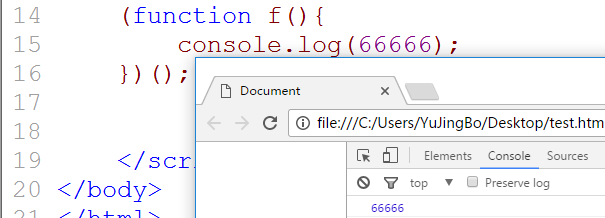
立即执行的表达式：

function(){} () 这样写对不对？

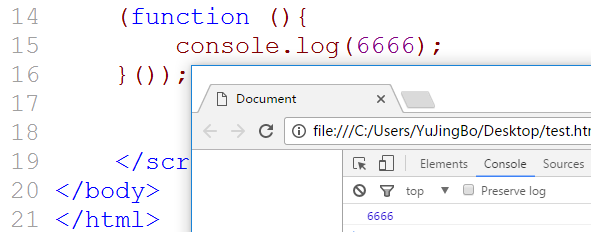
var f = function(){} 这里的function是一个表达式， funciton (){},此时这个function(){}是没有放到赋值运算符的右边，它不是一个表达式，它是一个语句。 是一个声明函数的语句。既然它是一个语句，语句后不能加（）; 如何去修改呢？

**实现的三种方式之一：**

（function(){}）(); 代码如下：



**实现的三种方式之二：**

（function(){}()）; 代码如下：

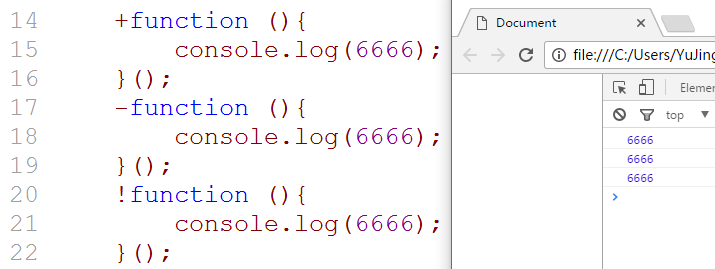
**实现的三种方式之三：**

+function(){}();

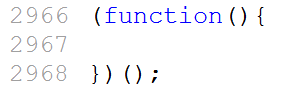
-function(){}();

!function(){}();

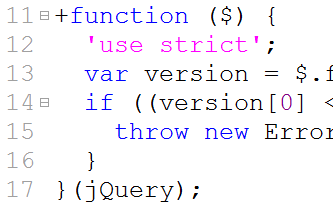
代码如下：



JQ中的应用：



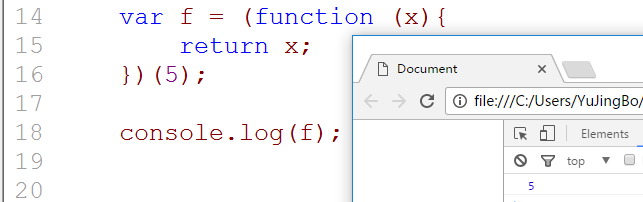
Bootstrap中的应用：



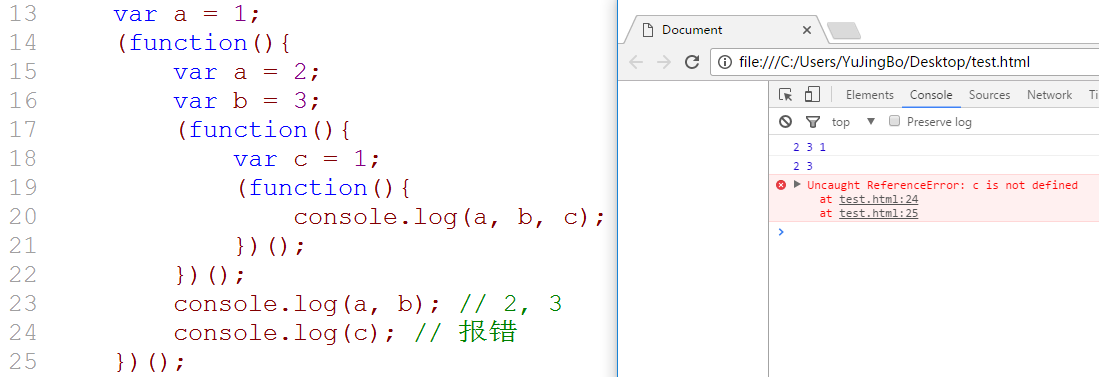
**还有一个知识点：**

带参数的立即调用函数表达式：

(function (形参)){}(实参);



将我们上午写的代码改成立即调用表达式：



**综上所述：立即调用函数表达式，可以模拟块级作用域，解决冲突（变量或函数名），不而不会去污染全局变量。**

注意的问题： 最后加一定要加分号（;）

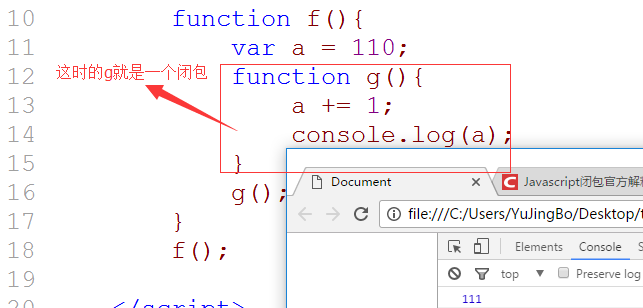
# 九，闭包

## 1，什么是闭包？

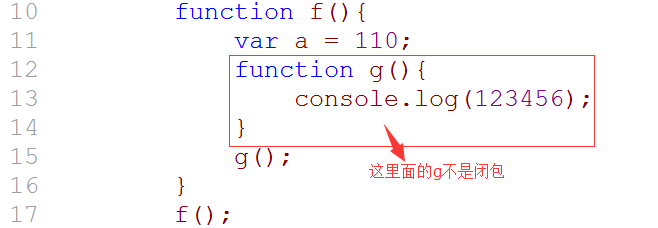
官方”的解释是：闭包是一个拥有许多变量和绑定了这些变量的环境的表达式（通常是一个函数），因而这些变量也是该表达式的一部分。

**理解闭包：**

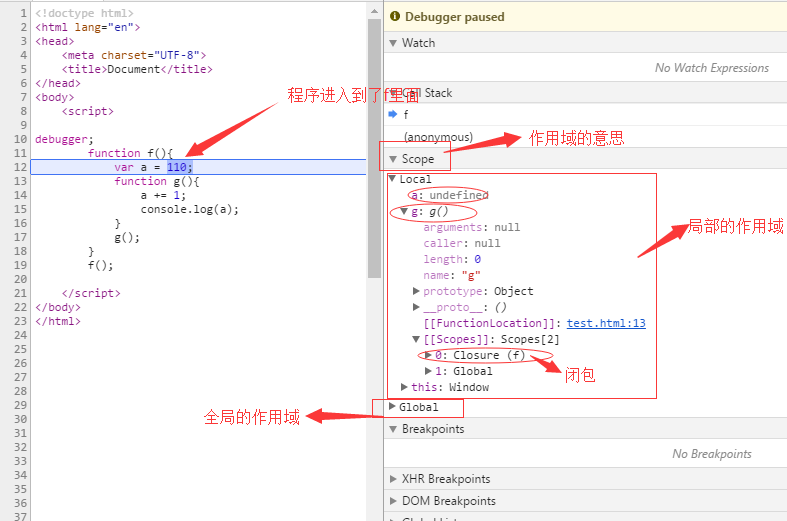
如果一个函数f内部定义了一个新的函数g, 并且这个函数g中引用了函数f中的变量，那么就称这个函数g是一个闭包。



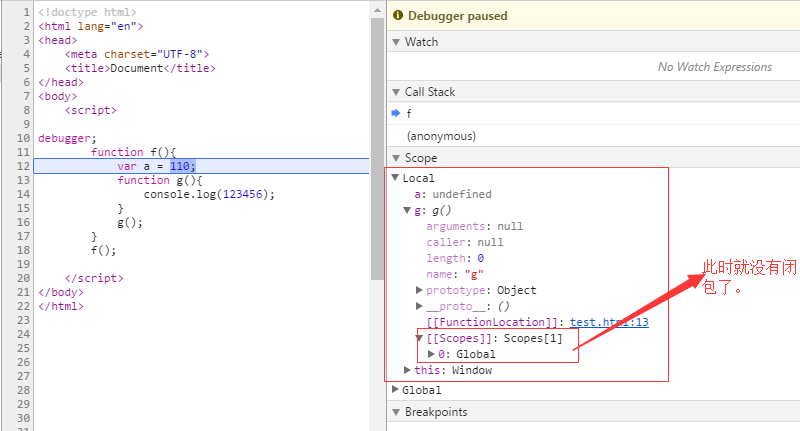
如果函数g没有引用函数f中的变量，那么这个函数g不能称为闭包：



使用工具查看如下：



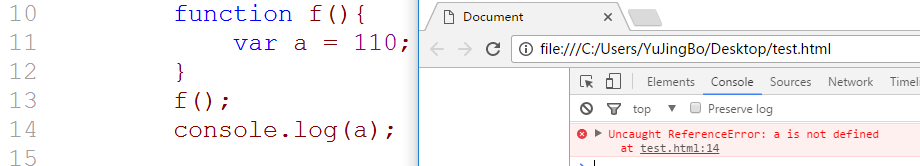
假如不让g引用f中的变量：那么g就不是一个闭包



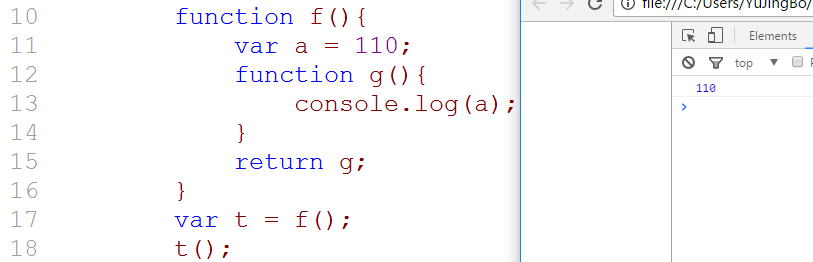
## 2，闭包的好处？ 延长变量的生命周期

我们前面说过：函数外面是不能访问函数里面的变量，但是闭包可以提供访问函数里面变量的机制。

在函数外面访问函数里面的变量，会出错：

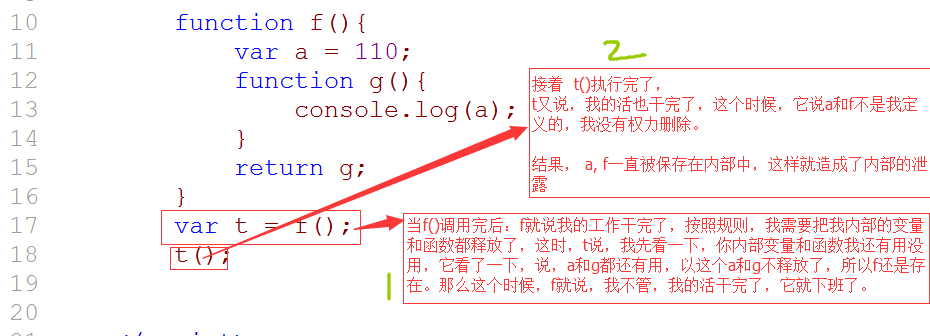


使用闭包： 我们使用闭包可以在函数外部去访问函数内部的变量



**意义：当一个函数调用完之后，那么这个函数中的变量就要被销毁。闭包可以去延长一个变量的生命周期。**

## 3，闭包的坏处？ 会造成内存泄漏。



**复习：**

1，执行上下文：JS代码分两种，全局代码，函数代码。当执行全局代码时就会产生全局的执行上下文，当执行函数代码时，就会产生函数的执行上下文。

2，执行上下文的作用：在代码执行过程中，需要的所有数据都有执行上下文提供。数据：变量和函数。

3，执行上下文件栈：入栈（函数调用），出栈（函数调用完）

4，执行上下文的组成部分：

函数的执行上下文：本函数内部的变量，函数，arguments + 父级函数的上下文

全局执行上下文：只有全局的变量和函数组成

5，函数的嵌套会形成一个作用域链

6，函数嵌套的原因：封闭性

7，函数是可以作为返回值

8，立即调用函数表达式IIFE， 原因：解决冲突，防止污染。

9，(function(){})(), (function(){}()), +/-/!funciton(){}()

10，闭包：如果一个函数A中，有另一个函数B，B引用了A中的变量，那么就称B是一个闭包。

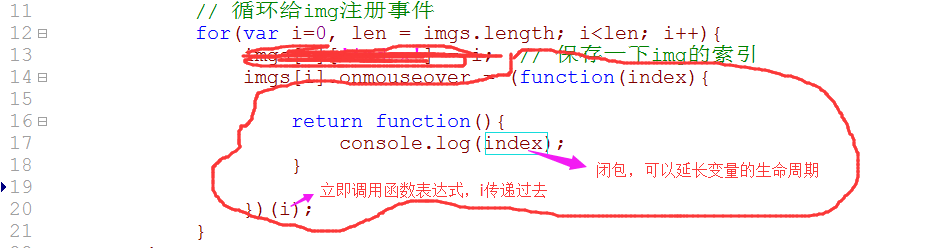
11，闭包的好处：延长变量的生命周期，不好的地方：造成了内存泄露。

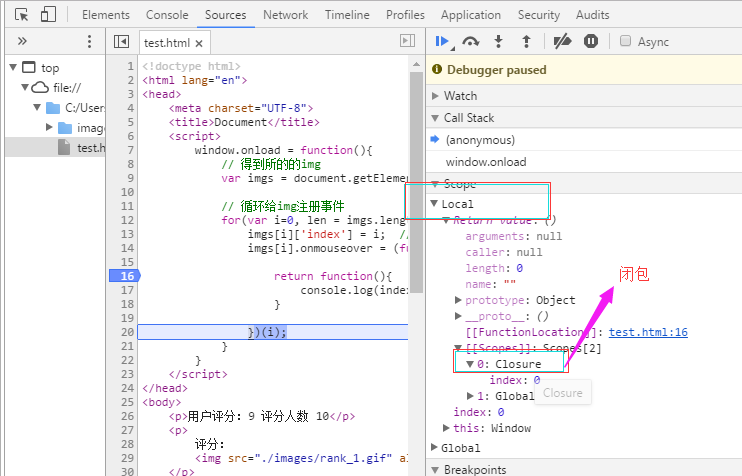
## 4，闭包的应用？

在小星星的案例中，我们是保存了img的索引：



**接着使用闭包，实现上面的代码：**





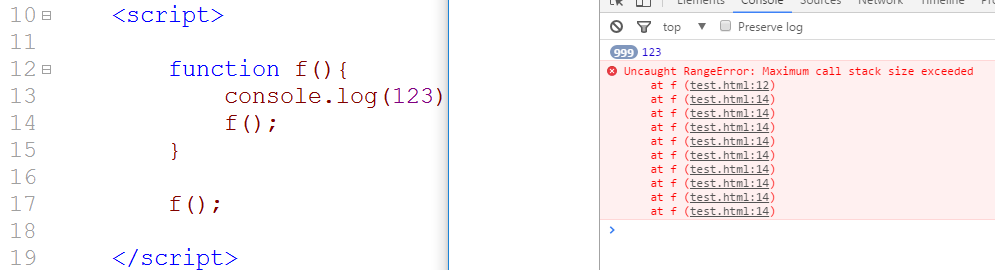
# 十，递归

## 1，什么是递归？ 理解递归？

程序调用自身的编程技巧称为递归（ recursion）。递归做为一种[算法](http://baike.baidu.com/item/%E7%AE%97%E6%B3%95" \t "http://baike.baidu.com/_blank)在[程序设计语言](http://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80" \t "http://baike.baidu.com/_blank)中广泛应用。 一个过程或[函数](http://baike.baidu.com/item/%E5%87%BD%E6%95%B0" \t "http://baike.baidu.com/_blank)在其定义或说明中有直接或间接调用自身的一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。

直白的说，就是函数在它内部调用了它自己，我们也称为递归函数。

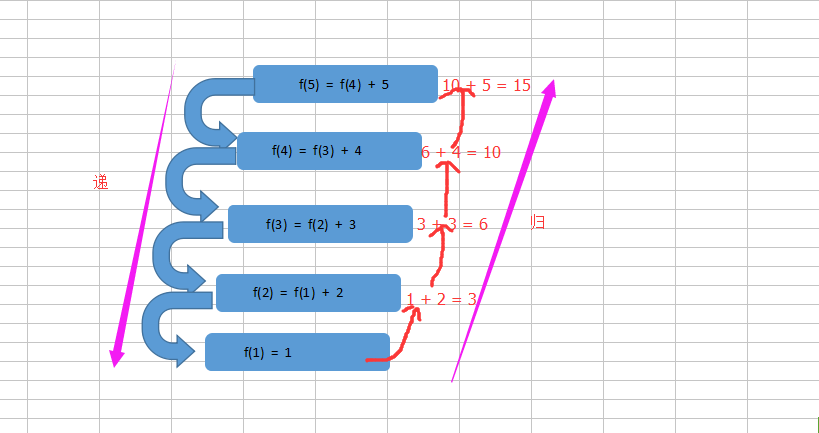
如下：



**举个例子：**

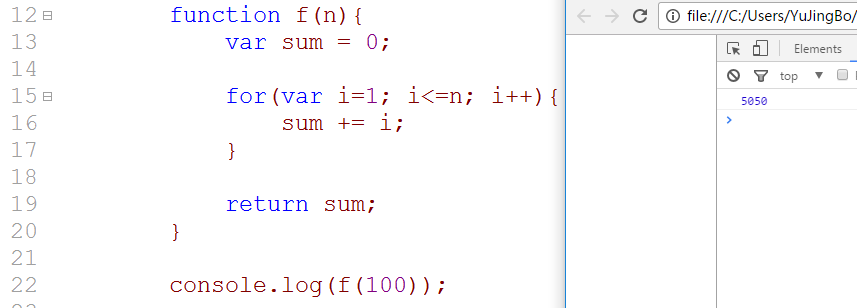
我要求5的累加：5 + 4 + 3 + 2 + 1 = 15。求一下，10000000的累加？

求5的累加， 先求4的累加 + 5， 对于4的累加，我们可以求3的累加+4， 对于3的累加，我们可以求2累加+3， 对于2的累加，我们可以求1的累加+2，对于1的累加就是1。 思路：大的问题，先化小，小的问题解决，再去解决稍微大点的问题.....最终最大的问题也解决。如下图：



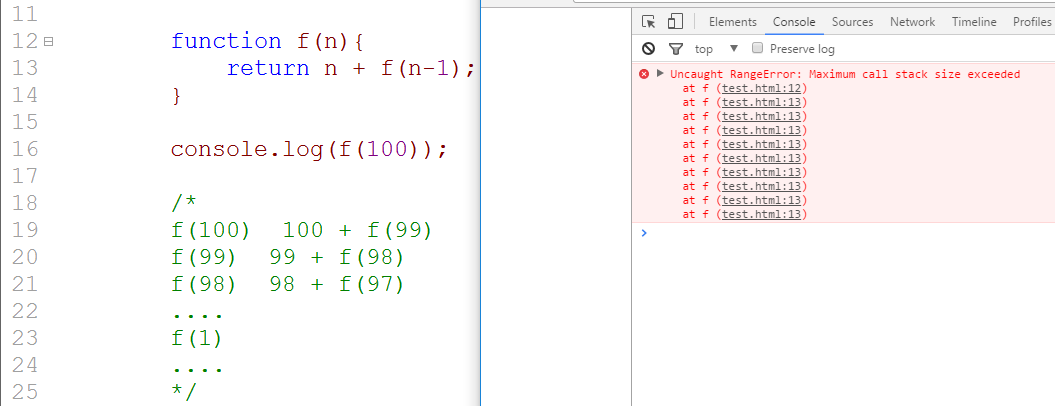
## 2，递归的应用？

### 求100累加？

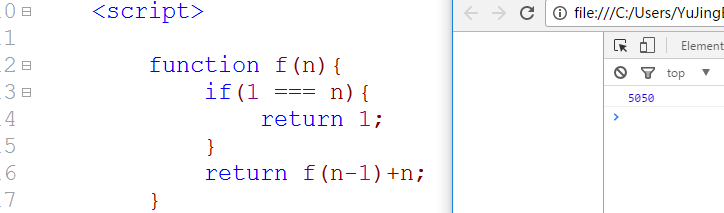
使用传统的方法：

使用递归：

使用递归时，一个要设置个结束条件，让它归，如果没有结果条件，就出错了：



**正确的写法：**



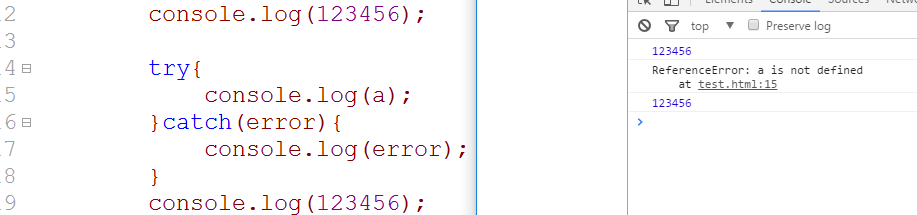
# 十一，异常处理

## 1，什么是异常？

异常不是错误！ JS中的错误分两种，一种是代码错误，一种是逻辑错误。

## 2，异常处理的语法结构？

try{}catch(error){}



## 3，异常处理的应用？