# 复习

for 循环

do while

后测试循环语句。直接执行循环语句，再进行出口判断

循环变量书写在循环语句外面

while

前测试循环语句。

循环变量书写在循环语句外面

break： 遇见break强制结束本层循环。

continue: 遇见continue结束本次循环。

函数：

|  |
| --- |
| 1. function 函数名(a,b) { 2. 语句; 3. } |

调用 函数名(实际参数，实际参数);

return返回值

# 函数

## 函数表达式

声明函数可以使用关键字function

还可以使用函数表达式

将一个匿名函数（拉姆达函数）赋值给一个变量。

|  |
| --- |
| 1. var 函数名 = function (形式参数) { 2. 语句； 3. }； //结尾必须添加分号 |

调用 函数名(实际参数);

|  |
| --- |
| 1. // 将匿名函数赋值给变量，结尾必须添加分号 2. var fun = function (a,b) { 3. return a + b; 4. }; 5. // a,b数字 字符串 6. // 调用函数名() 7. alert(fun(1,10)); 8. console.log(fun("1","你好")); |

## 函数数据类型

简单数据类型：number, string, undefined, boolean

引用数据类型：object, function

不管是关键字function还是函数表达式数据类型都是function。

简单数据类型：不同变量在赋值，变量将保存的值复制一份然后进行赋值。变量改变不互相影响

|  |
| --- |
| 1. //简单数据类型 2. var a = 10; 3. // b = 10 4. var b = a; 5. // 改变a 6. a = "你好"; 7. console.log(b); //b=10 |

引用数据类型：不同变量在赋值，将变量保存的地址进行赋值，变量改变会互相影响

|  |
| --- |
| 1. // 引用数据类型 2. var fun1; 3. function fun2() { 4. return 10; 5. } 6. // fun2地址指向了fun1 7. fun1 = fun2; 8. // 改变 9. fun2.xixi = "嘻嘻"; 10. fun1.haha = "哈哈"; 11. console.log(fun1.xixi); 12. console.log(fun2.haha); |

总结：简单数据类型保存的值，引用数据类型保存的地址。

## 函数声明提升

变量提升只提升声明语句，赋值语句不提升。

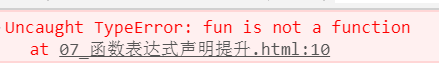
function关键字声明函数提升。在函数预解析时将函数名提升到所有语句之前，由于函数保存的是地址，通过地址可以找到声明整体。相当于函数整体提升到所有语句之前。

函数先调用，在声明也不会报错。

|  |
| --- |
| 1. // 先调用 2. fun(); 3. // 后声明 4. function fun() { 5. console.log(1); 6. } |

函数表达式，提升只提升声明语句，赋值语句不提升。先调用会报错。

|  |
| --- |
| 1. // 先调用 2. fun(); 3. // 后声明 4. var fun = function () { 5. console.log(1); 6. }; |



总结：在声明函数时最好使用function关键字声明，不会出错。

function声明函数名和变量相同，优先提升function，给function使用。

|  |
| --- |
| 1. // 先使用 2. console.log(fun); 3. // 使用function声明fun函数 4. function fun() { 5. return 1; 6. } 7. // 声明fun变量 8. var fun = 10; 9. /\* 10. 预解析： 11. 优先提升函数 12. function fun(){};名字不会重复声明 13. console.log(fun); 14. fun = 10; 15. \*/ 16. console.log(fun); |

函数表达式和function关键字重名，优先给function使用。

|  |
| --- |
| 1. // 先使用 2. fun(); 3. // 声明函数表达式 4. var fun = function () { 5. console.log(1); 6. }; 7. // function关键字 8. function fun() { 9. console.log(2); 10. } 11. /\* 12. 预解析：优先提升function 13. function fun(){console.log(2)}; fun名字声明过不会重复声明 14. fun(); 15. fun = function() {console.log(1)}; 16. \*/ 17. fun(); |

## 1.4递归函数

在函数内部调用自身。一般解决数学问题

菲波那切数列：

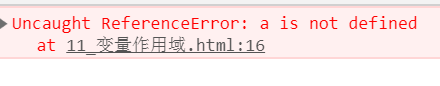
1,1,2,3,5,8,13,21……

|  |
| --- |
| 1. /\* 2. 1,1,2,3,5,8,13,21…… 3. 函数：用户输入一个项数，告诉用户该项对应的值 4. 从第3项开始，该值=前1项+前2项对应值 5. 参数： 项数1- 6. 返回值：该项对应的值 7. \*/ 8. function feibo(n) { 9. // 求第n项对应的值 10. if (n === 1 || n === 2) { 11. return 1; 12. }else { 13. return feibo(n-1) + feibo(n-2); 14. } 15. } 16. console.log(feibo(1)); 17. console.log(feibo(2)); 18. console.log(feibo(3)); 19. console.log(feibo(4)); 20. console.log(feibo(8)); 21. console.log(feibo(20)); |

## 1.5变量作用域

在函数内部声明的变量只能在函数内部使用，在函数外部任何地方都不能使用。

|  |
| --- |
| 1. // 在函数内部声明a变量 2. function fun() { 3. var a = 10; 4. console.log(a); 5. } 6. // 在函数外部使用a 7. console.log(a); |



js中只有函数可以关住变量的作用域。

## 1.6 局部变量和全局变量

局部变量：在一个作用域（定义域）内定义的变量就是这个作用域内的局部变量。只能在作用域内被访问到。

全局变量：从广义上来看，全局变量也是一种局部变量。全局变量定义在全局，所以也叫全局变量。可以在任何地方都被访问到。

变量申明的原理：全局变量，在全局定义之后，会永久存在，任何时候，任何位置访问，都能够找到它。局部变量定义在函数内部的，函数定义的过程，并没有真正的去定义这个局部变量，只有在执行函数的时候，才会立即定义这个局部变量，执行完之后，变量就被立即销毁了，在其他的地方访问变量的时候，找不到这个变量，所以会有一个引用错误，变量未定义。

|  |
| --- |
| 1. // 没有书写在函数内部的变量，可以在全局范围内访问 2. var b = "你好"; 3. // 在函数内部声明a变量 4. function fun() { 5. // a只能在函数内部使用，a局部变量 6. var a = 10; 7. console.log(a); 8. // 全局b，可以在全局使用，当然可以在函数内部使用 9. console.log(b); 10. } 11. // 在函数外部使用a 12. // console.log(a); 13. fun(); 14. console.log(b); |

## 1.7 形参是局部变量

形参只能在该函数内部使用，在函数外部不能访问，会报错。

|  |
| --- |
| 1. function fun(a,b) { 2. console.log(a); 3. console.log(b); 4. } 5. fun(1,10); 6. // 形参是fun()函数局部变量，在fun函数外部不能使用 7. console.log(a); 8. console.log(b); |

## 1.8 全局变量使用

函数间的通信作用：函数声明一次可以多次调用，全局变量可以进行累加。

|  |
| --- |
| 1. // 声明全局变量a 2. var a = 1; 3. // 声明函数 4. function jia() { 5. a ++; 6. console.log(a); 7. } 8. // 函数间的通信，都可以改变a 9. jia(); //a=2 10. jia(); //a=3 11. console.log(a); |

全局变量作用：传递作用

不同的函数都可以改变全局变量值，并且使用最新值参与计算。（信号量）

|  |
| --- |
| 1. // 声明全局变量a，信号量 2. var a = 1; 3. // 声明函数1让a自加 4. function jia() { 5. a ++; 6. console.log(a); 7. } 8. // 声明函数2让a自减 9. function jian() { 10. a --; 11. console.log(a); 12. } 13. jia(); //a=2 14. jian(); //a=1 15. jia(); 16. jia(); 17. jia(); //a=4 18. console.log(a); |

## 1.9 作用域链

指的是我们变量查找的一个规律：我们可以在不同的作用域内使用相同的标识符去命名变量。我们在使用一个变量的时候，需要找到匹配的标识符，我们有重复的，用哪一个？如果在当前作用域有这个变量，就直接使用，如果当前作用域没有这个变量定义，会一层一层的从本层往外依次查找，遇到第一个就直接使用。类似于就近原则。

|  |
| --- |
| 1. // 声明全局a 2. var a = 1; 3. function fun1() { 4. // 声明fun1局部变量a 5. var a = 2; 6. // a=2 7. console.log(a); 8. function fun2() { 9. var a = 3; 10. function fun3() { 11. // 本层fun3没有a定义；从本层出发依次向外查找。 12. //a=3 13. console.log(a); 14. } 15. fun3(); 16. } 17. fun2(); 18. } 19. fun1(); 20. console.log(a); |

当遇见一个变量时，JS引擎会从其所在的作用域依次向外层查找，查找会在找到第一个匹配的标识符的时候停止。在多层嵌套的作用域中可以定义同名的标识符，发生“遮蔽效应”。

如果变量声明时，不写var关键字，计算机会自动在全局作用域内给它进行一个声明，局部变量就强制性的变成了全局变量。这种情况是不合理，会造成一个全局变量的污染。所以，定义变量必须写var关键字。

|  |
| --- |
| 1. // 声明全局a 2. var a = 1; 3. // a=2; a=3 4. function fun1() { 5. a = 2; 6. console.log(a); 7. function fun2() { 8. a = 3; 9. function fun3() { 10. console.log(a); 11. } 12. fun3(); 13. } 14. fun2(); 15. } 16. fun1(); 17. // 由于变量强制转为全局变量a=3 18. console.log(a); |

## 1.10 函数作用域

函数的作用域和变量作用域相似，也是函数只能在声明函数的内部使用，在声明函数外部不能使用。

|  |
| --- |
| 1. function outer() { 2. var a = 1; 3. // inner函数在outer内部声明，只能在outer内部使用 4. function inner() { 5. console.log(a); 6. } 7. inner(); 8. } 9. outer(); 10. // 在outer外部不能使用inner 11. inner(); |

总结：函数能关住变量和函数的作用域。

## 1.11 闭包

函数可以记住声明时所处的外部环境和内部语句，把这个整体称为闭包（密闭整体）。不管函数在哪里调用都会参考

|  |
| --- |
| 1. function outer(){ 2. var a = 1; 3. function inner(){ 4. console.log(a); 5. } 6. return inner; 7. } 8. var inn = outer(); 9. inn(); |

inner函数把它自己内部的语句（console.log(a)），和自己声明时所处的作用域(var a = 1;)一起封装成了一个密闭环境，我们称为“闭包”。



闭包天生存在，并不需要什么特殊的结构才存在，只不过我们必须要刻意地把函数放到其他的作用域中调用，才能明显的观察到闭包性质。

* 闭包外部环境并不是一成不变的。

|  |
| --- |
| 1. function fun1() { 2. var a = 1; 3. function fun2() { 4. a ++; 5. console.log(a); 6. } 7. return fun2; 8. } 9. /\* 10. 对于fun2 外部环境a=1 内部语句a++; console.log(a); 11. \*/ 12. var inn = fun1(); 13. inn(); //a=2 14. inn(); 15. inn(); 16. inn(); 17. inn(); 18. inn(); |



* 函数每一次调用都产生一个全新的闭包，外部环境和内部语句都是全新的

|  |
| --- |
| 1. function outer(a) { 2. function inner(b) { 3. console.log(a + b); 4. } 5. return inner; 6. } 7. /\* 8. 第一个包 9. outer(10) 初始定义inner 外部环境a=10 和 内部语句console.log(10 + b) 10. \*/ 11. var fun1 = outer(10); 12. /\* 13. 第二个包： 14. 外部环境a = 100 15. 内部语句console.log(100 + b) 16. \*/ 17. var fun2 = outer(100); 18. fun1(1); 19. fun2(1); |

# 数组

## 2.1 概述

数组：有序的数据集合。array

数组字面量： []

数据每一项之间用逗号隔开，最后一项不要书写逗号。

数组中数据可以是相同数据类型，也可以是不同数据类型

|  |
| --- |
| 1. // 数组可以是相同数据类型 2. var arr1 = [1,2,3,4,5,6,7,8,8]; 3. // 还可以是不同的数据类型 4. var arr2 = [10,"你好",undefined,true,"false",function(){console.log(1)}]; 5. console.log(arr1); 6. console.log(arr2); |

读取：通过索引值index,索引值从0开始

|  |
| --- |
| 1. arr[index] 2. arr2[0] |

设置：=

|  |
| --- |
| 1. arr1[0] = "我是开头值"; |

## 2.2 length属性

数组保存数据的总个数。

数据类型：object

|  |
| --- |
| 1. object.length |

数组最后一项对应的索引值可以使用length表示

|  |
| --- |
| 1. arr.length - 1 |

数组可以通过Length将数组强制拉长，中间没有赋值，数据是undefined

|  |
| --- |
| 1. arr2.length = 10; |

数组可以通过索引值强制拉长，没有赋值仍然是undefined

|  |
| --- |
| 1. // 索引值也可以赋值，将数组拉长 2. arr2[20] = 20; 3. console.log(arr2.length); 4. console.log(arr2[18]); 5. console.log(arr2); |

## 2.3 数组遍历

|  |
| --- |
| 1. // 遍历索引值从0开始到arr.length - 1 2. for (var i = 0; i <= arr.length - 1; i ++) { 3. console.log(arr[i]); 4. } |

# 数组方法

## 数组首尾操作的方法

push(): 在数组结尾添加数据

参数：添加的数据，可以有多项，用逗号隔开

返回值：新数组长度

pop(): 删除数组最后一项。

参数：不用

返回值：数组最后一项对应的值

unshift(): 在数组开头添加数据

参数：添加的数据

返回值：新数组长度

shift(): 删除数组第一项

参数：不用

返回值：删除项

## 数组合并方法

concat(): 用于数组合并

参数：变量，字面量，散的值

返回值：合并后新数组

原数组不改变

## 数组截取方法

slice(): 用于数组截取

参数：第一个参数start,第二个参数end

返回值：截取到的新数组

原数组不改变

|  |
| --- |
| 1. arr1.slice(1,4); |

参数：还可以使用负数，表示倒数，从-1开始，从小索引值到大索引值

|  |
| --- |
| 1. arr1.slice(-5,-2) |

可以省略第二个参数，表示截取到最后

|  |
| --- |
| 1. arr1.slice(1) |

## 数组删除，插入，替换方法

splice(index, howmany, elements):

参数：

index:删除项对应的索引值

howmany:删除元素的个数

elements:插入的数据

返回值：删除数据组成数组

原数组改变

删除功能：index，howmany不能是0都必须书写，elements省略

插入功能：index插入项索引值，howmany必须是0都必须书写，elements书写要插入数据

替换功能：index，howmany必须不是0，elements书写要替换数据

## 倒序方法

reverse(): 倒序，只能将数组前后顺序进行颠倒，第一项变为最后一项。

参数：不用

返回值：倒序之后新数组

原数组改变

## 排序

sort():

参数：不用

返回值：排序之后新数组，原数组改变

排序原理：将数组每一项转为字符串，按照字符编码进行排序。

数字，大写字母，小写字母

|  |
| --- |
| 1. var arr2 = [23,8,"black",true,false,undefined,"B"]; 2. // 排序 3. console.log(arr2.sort()); 4. console.log(arr2); |



参数：自定义匿名函数（比较函数）

形参a,b表示两个要比较的数字

自定义升序排序：

如果a>b，返回值是正数。排序之后a应该在b后面

如果a ===b，返回值是0.排序之后a，b不区分前后

如果a<b,返回值负数。排序之后a应该在b前面

|  |
| --- |
| 1. var arr = [10,20,45,8,9,55,45]; 2. // 自定义升序排序 3. arr.sort(function (a,b) { 4. if (a > b) { 5. return 3; 6. }else if (a === b) { 7. return 0; 8. }else { 9. return -2; 10. } 11. }); 12. // 原数组改变 13. console.log(arr); |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |