# 函数\*\*\*\*\*\*\*\*\*\*

## 函数基本使用

### 1、概念，定义（表达式，字面量），作用

1）什么是函数：

具有某种特定功能的代码块~ //想象成工具或者工厂

函数其实本质也是一种数据，也是属于对象数据类型；

2）为什么要有函数

1、解决代码的冗余问题，形成代码复用；

2、可以把整个代码项目，通过函数模块化；

3、封装代码，让函数内部的代码对外部不可见；

3）函数引入

打印等腰三角形

4）函数定义（字面量定义和表达式定义 先定义后使用 三要素\*\*\*）

字面量定义：

function 函数名（）{

代码块；（函数体）

}

函数表达式定义：

var 变量名（函数名） = function(){

代码块；（函数体）

}

5）函数调用

注意：函数的使用，必须有函数定义和函数调用两部分；函数调用了才会去执行函数定义的代码；

函数的定义（三要素）：

功能（函数名，见名思意，看到了函数名就想到了这个函数的功能，函数名字最好使用动词）

参数：（）里面是函数的参数，定义函数的时候要考虑这个函数是否有参数；

返回值：函数最终都会有一个返回值；定义函数的时候也要考虑这个函数是否需要返回值；

函数的调用：

本质上是函数调用表达式；这个表达式整体的值就是函数定义当中return后面的值；如果函数定义当中没有写return，那么其实相当于默认return的是一个undefined.只是把return省略了；return关键字在函数当中一是用来返回值，二是当函数执行到return的时候，函数就立即结束了，即使return下面还有代码，也不再执行；

参数：函数定义的时候我们要考虑的三要素包含了一个参数；参数在函数定义当中被称作形式参数，简称形参；

函数调用的时候，我们在（）当中所写的也叫做参数，叫做实际参数，简称实参； 函数调用的时候，我们写实际参数被称作函数传参，传参的本质是把函数调用的实参，赋值给函数定义的形参； 函数的实参，肯定是一个有值的东西；函数定义的形参：本质上相当于在函数内部var了变量；

## 函数的分类

无参无返回值 //通常情况下 很少有这样的函数，一般都是需要将代码模块化的时候，纯粹的去封装功能使用的；

有参无返回值

无参有返回值

有参有返回值

## 强化练习

编写求1到n的和函数

编写函数实现求一个数的阶乘

编写函数求数组的最大值，最小值

封装函数加工数组，每一项加10输出

封装函数实现打印1到N的质数;

封装函数实现对数组进行排序；

封装函数实现对数组翻转

封装函数实现对数组去重

\*\*\*\*\*\*函数的调用过程（内存角度）

## 作用域

### 1、作用域概念，定义，作用，分类

### 2、全局变量和局部变量

### 3、作用域链

### 4、案例:

面试题

var num = 10;

function fun() {

var num = 20;

fun2();

}

function fun2() {

console.log(num);

}

fun();

### 特别专题（画图相当重要）：

#### 1）内存堆栈

内存当中有两个重要的结构：栈和堆

栈：栈结构先进后出（FILO） 可以想象成一个桶 栈结构当中开辟的内存，比较小，速度比较快，操作系统自动分配自动回收；

堆：堆结构是链表结构，可以随意的在任何位置添加和删除；可以想象成一个链条； 堆结构当中开辟的内存，比较大，速度比较慢，

一般都是由程序员自己用的时候再去申请分配；要想让内存回收，必须手动释放回收；

在js当中，我们的内存回收几乎都是计算机自动回收；因为js里面封装了垃圾回收机制；

访问一个对象：

var arr = [1,2,3];

console.log(arr) 打印出来的是数组数据

var arr2 = arr; //arr2当中存储的是数组的地址，代表着arr和arr2同时指向了同一个数组；

arr[0] = 100;

console.log(arr2);//[100,2,3];

#### 2）程序开始执行到结束都做了什么：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1、程序一开始执行，碰见了全局环境，首先会创建全局环境并且进行压栈，全局代码执行的时候依赖的就是全局环境当中的东西；比如 全局变量（全局变量如果存的是基本数据类型，那么这个值是直接存在栈当中的，如果这个变量存的是对象类型（函数、数组），那么数据是要在堆结构当中开辟自己的空间专门存储的。然后把堆里面这块空间的地址存给栈当中的对应变量）；

2、当程序执行碰到了函数调用；函数是比较特殊，因为它也可以执行；函数执行的时候也要有自己的环境去依赖。因此函数执行也是创建自己的函数环境进行压栈（函数环境一定是压在全局环境之上的），局部变量，是在函数环境当中存在的，只有函数执行，局部变量才会出现。函数执行完成以后，函数环境要弹出栈（销毁归还内存）,局部变量也就不复存在了。

3、当函数调用完成以后，会继续执行全局代码，一直到所有的代码都执行完成，代表程序执行结束，程序结束的时候，我们的全局 环境最后出栈。

## 预解析

a) 预解析演示

b) 预解析

程序在代码执行之前会先进行预解析；

预解析会解析带var的变量和函数声明定义的函数

解析函数优先级比解析变量要高：可以认为解析的时候分为两步，先去解析所有的函数，再去解析所有的带var变量；

解析过程当中，如果函数重名，会覆盖（后面的函数会把前面的覆盖掉）

如果变量重名，会忽略；

解析函数的时候，函数定义方式不同，解析过程也不大一样：

如果是字面量定义，整个函数都要被提升

function func(){};

如果是表达式定义，只会提升变量

var func = function(){};

总结：预解析先去解析函数声明定义的函数，再去解析带var的变量；

函数重名会覆盖，变量重名会忽略；（变量如果不带var，变量是不会进行预解析的；只有带var的变量才会进行预解析；表达式定义的函数也是当做变量去解析）

c) 面试题:

--------------------------------------------------------

alert(a);

a = 0;

--------------------------------------------------------

alert(a);

var a = 0;

alert(a);

--------------------------------------------------------

alert(a);

var a = '我是变量';

function a(){ alert('我是函数') }

alert(a);

--------------------------------------------------------

alert(a);

a++;

alert(a);

var a = '我是变量';

function a(){ alert('我是函数') }

alert(a)

--------------------------------------------------------

alert(a);

var a = 0;

alert(a);

function fn(){

alert(a);

var a = 1;

alert(a);

}

fn()

alert(a);

--------------------------------------------------------

alert(a);

var a = 0;

alert(a);

function fn(){

alert(a);

a = 1;

alert(a);

}

fn()

alert(a);

--------------------------------------------------------

以后我们遇到程序

1、先把代码调整好（预解析,把该提升的提升好）;

2、画图 （全局环境）

3、执行全局代码

4、碰到函数调用先把函数内部的代码调整好（函数内部预解析，把该提升的提升好）；

5、画图（函数环境）

6、执行函数内部代码

7、函数死，继续执行全局代码

8、全局死

## IIFE，回调函数 函数递归 arguments

### 1、IIFE:

Immediately Invoked Function Expression意为立即调用的函数表达式，也就是说，声明函数的同时立即调用这个函数。

语法：

(function(){

代码块；

})();

特点：

函数定义的时候同时执行

只执行一次

不会发生与解析（函数内部执行的时候会发生）

作用：

防止外部命名空间污染

隐藏内部代码暴露接口

对项目的初始化

### 2、Arguments 函数实参伪数组

### 3、回调函数：函数是我定义的 我没有调用 最终执行了

事件

定时器

ajax

生命周期回调函数

### 4、函数递归

作业:

面试题: 函数，作用域，预解析（必须理解）