# 数组和函数 \*\*\*\*\*\*\*\*\*\*

**一：数组**

### 1、数组概念，作用，定义，基本操作

1.是什么：具有相同类型（或者不同类型）的数据有序集合；

数组也是一种数据，它是属于对象数据类型；

2.为什么：

一次性的让我们存储多个数据

3.数组的定义：

1、字面量定义：

2、构造函数定义 坑：如果小括号当中只写了一个数字，代表的是定义的数组长度；

4.数组length，索引（下标）

只要定义一个数组，数组里面就会有一个默认的属性叫length,它代表着数组的长度

索引也被称作下标，通常情况下只要我们知道了索引，就可以拿到这个数组对应的这个索引的值

当定义完一个数组的时候，数组的下标最大值是数组的长度 - 1；

5.数组的操作

数组可以通过下标去操作元素(增 改 查)

数组的length可以操作数组添加和删除

数组的遍历

### 2、数组案例

数组求和

求数组最大值，最小值，平均值

合并数组

反转数组

数组去重

冒泡排序

### 3、二维数组（了解）

## 二、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）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）

}

构造函数定义:

var fn3=new Function(“代码体”)

1. 函数三要素

6）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；

### 基础练习

封装函数打印100遍我爱你

封装函数打印n遍你想要打印的内容

封装函数返回2+3的和

封装函数返回n+m的和

### 3、强化练习

编写函数返回1到n的和

编写函数返回一个数的阶乘

编写函数实现返回某个数组的最大值，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;

封装函数返回对数组排序后的结果；

封装函数返回对数组翻转后的结果

封装函数返回对数组去重后的结果

## 作用域

### 作用域概念，作用，分类

1. 什么是作用域： 变量起作用的范围区域
2. 作用：隔离变量
3. Es5当中只有全局作用域和局部作用域（函数为界）
4. 作用域是虚拟的概念，代码写完作用域就存在（函数定义完）

### 全局变量和局部变量

var a =10;

    function fn(){

      var b=20;

      console.log(a)

      console.log(b)

}

fn()

    console.log(a)

    console.log(b)

1. 全局变量就是在全局作用域当中定义的变量
2. 局部变量就是在局部作用域当中定义的变量
3. 全局变量整个程序都能操作，函数内外部都可以看到都能使用
4. 局部变量只能在函数内部操作
5. 如果全局变量和局部变量重名，各自是各自

var a = 10;

    var b = 20;

    var c = 30;

    var d = 40

    function fn(){

      var a = 100;

      var b = 200;

      var c = 300;

      var d = 400;

      // d = 400;

      // e = 500;

      console.log(a,b,c,d)

    }

    fn()

    console.log(a,b,c,d)

如果内外变量都带var,外部和内部各自使用各自

如果内部碰到不带var的变量，按以下步骤去走：

当函数内部出现变量不带var

先看函数内部有没有定义var这个变量，如果有就是局部变量

如果函数内部没有定义var这个变量，找形参，形参如果有，当做局部变量处理

如果形参也没有，那么找全局，全局有，那么就是在操作全局变量

如果全局也没有，那么相当于在全局定义了一个变量（全局变量）

### 作用域链

var a = 0;

    function fn1(){

      var a = 1;

      function fn2(){

        var a = 2;

        function fn3(){

          var a = 3;

          console.log(a)

        }

      }

    }

和作用域完全不是一个东西

作用域链描述的是程序查找变量的过程

首先在自己的作用域当中去查找，如果查找不到，去到上级作用域去查找，查找就用，查不到继续往上查找，直到找到真正的全局，找到就用，找不到报错（引用错误，这个变量没定义）

作用域链的顶端 一定是全局；

作用域链是真实存在的，我们后期是可以看到的

作用域是函数定义好就存在的，而作用域链是函数调用的时候才有的；

### 4、案例:

面试题

var num = 10;

function fun() {

var num = 20;

fun2();

}

function fun2() {

console.log(num);//?

}

fun();

//函数定义时候作用域就定死了，调用的时候作用域链顺着作用域去查找，函数的作用域和调用没关系

## 四、预解析

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());

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);

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

alert(a);

var a = 0;

alert(a);

function fn(){

alert(a);

a = 1;

alert(a);

}

fn()

function fn(){

alert(a);

var a = 1;

alert(a);

}

var fn = function(){

alert(a);

a = 1;

alert(a);

}

fn();

var a = 100;

alert(a);

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

## 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：栈内存和堆内存

栈：栈内存先进后出（FILO） 栈内存当中开辟的内存，比较小，存储速度比较快；

堆：堆内存是链表结构， 堆内存当中开辟的内存，比较大，速度比较慢；

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

题目：

1. var a = 10;

var b = 20;

function add(a,b){

a = 40;

return a + b

}

add(a,b)

console.log(a,b)

2.var arr = [1,2,3,4]

var a = 10;

var b = 20;

function fn(arr){

var a = 100;

b = 200;

console.log(a);

console.log(b);

}

fn(arr);

console.log(arr);

console.log(a);

console.log(b);

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），那么数据是要在堆内存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。

3）基本数据类型和引用数据类型区别（画图）：

1.

var a = 10;

      var b = a;

      a = 20;

      console.log(b);

2.

var arr1 = [1,2,3];

      var arr2 = arr1;

      arr2[1] = 22;

      console.log(arr1);

3.

var arr1 = [1,2,3];

      var arr2 = arr1;

      arr2 = [1,22,3];

      console.log(arr1);

4.

var arr1 = [];

      var arr2 = [];

      var arr3 = arr1;

      arr3 = [];

      arr3[0] = 12;

      console.log(arr1,arr2,arr3);

## 五、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

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

### 函数递归

function fn(x){

    if (x == 1) {

        return 16

    }

    return fn(x-1) - 2

}

fn(3)