1、**浏览器加载HTML页面的时候**，首先会**提供**一个全局**js代码执行的环境：全局作用域**

2、**预解释** （**变量提声**） **浏览器天生自带**

当前的作用域中，**js代码执行之前**，**浏览器**首先**会默认把**所有的**带var**和**function**的**进行提前的声明或者定义**

**声明（declare）：**var num；**告诉**浏览器在**全局作用域**中**有**一个全局的**变量了**

**定义（defined）：**num=12；给我们的变量进行**赋值**

**var和function**

**var**的变量**提前声明**（预解释）但是**不会赋值** 但有一个**默认的undefined值**，当**代码执行**的时候**才会赋值**。

**function**的会**提前声明**并**赋值**

**预解释只发生**在**当前**的**作用域下**，例如：**开始只对windows下**的进行**预解释**，只有**函数执行**的时候**才会对函数中**的进行**预解释**

**js中内存的分类：**

**栈内存：代码运行的环境**也是作用域：作用域（全局作用域/私有作用域）**基本数据类型都 存在栈内存里**

**堆内存：**用来**保存值，存储引用数据类型的值**：对象存储的是属性名和属性值，**函数存储的是代码字符串...**

console.log(num); //undefined  
console.log(obj); //undefined  
console.log(sum); //function本身  
var num=12; //赋值  
console.log(num);  
var obj={'name':'zqb',age:30};  
console.log(obj);  
console.log(haha); // 报错haha is not defined  
var a =function(){}; //不会提前赋值，**预解释只会看等号左边的变量**，**并不会看你的值 是什么** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*无节操  
function (){ //当代码执行的时候，声明和赋值已经结束了，直接跳过  
 function haha(){} //我们的**预解释只发生在当前作用域\*\*\*\*\*\*\*\*\*\*\*\*\***  
 var total=0;  
 total=num1+num2;  
}  
  
**加var和不加var的区别：（**在全局作用域下**）**

1**是否被提前声明**

2**不加var**就是**赋值**，相当于给**window添加了一个属性**

当函数运行的时候就会产生一个私有的作用域，并且这个变量也是私有的变量。并且这个私有变量在面访问不到，我们把这种机制叫做**闭包**。私有作用域，不收外界干扰。

**函数的运行：（顺序）**

1如果有形参是**形参赋值** 2**预解释** 3**代码**逐行**执行**。。

**区别私有变量还是全局变量：（逐级往上查找）**

如果**带var**就是**私有变量**，如果是**形参**，**也**可以**理解是一个私有变量**。**代码在执行的时候，首先查找**当前环境内的（作用域，栈）的**私有变量**，如果有就直接用，如果没有**去上一级作用域去查找**，如果**有就拿来用**，如果**没有一直查找**到**顶级的window全局作用域**，如果**没有就报错了，not defined**，我们把**这种查找机制叫做作用域链**  
function b(){  
console.log(total); //undefined  
var total=0;  
console.log(window.total); //0  
function sum(num1,num2){  
 console.log(total); //undefined  
 var total=num1+num2;  
 console.log(total); //30  
}  
sum(10,20);  
console.log(total); //0  
}  
b();

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function fn(){  
 total=100; //赋值的过程，直接赋值给widow  
 //var total-100 这个私有变量和外面没有任何关系 加var 闭包 报错  
 }  
fn();  
console.log(total); //100  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var num=12;  
 function fn(){  
 var num=100;  
 return function {  
 console.log(num);  
 }  
 }  
 var f=fn(); //函数的执行 赋值是执行的结果 没有return就是undefined  
 fn();  
 //函数运行时候的上一级作用域只和在哪里定义有关系  
  
**预解释的时候**  1**不论条件是否成立都会预解释**（if）  
 2**预解释**的时候**带var关键字**的**只看等号左边**  
 3**全局作用域下**的**自执行函数不会预解释**，预解释**只发生**在**当前作用域**  
 4**return后面**的**代码**仍然**会被预解释**，但是**return出来**的**值不会被预解释**， **即使是一个函数也不会被预解释**

//1 if('tota' in window){  
 var total =6;  
 }  
 console.log(total);//6  
//2 var a=function(){  
 alert();  
 }  
//3 console.log(b);  
 !function(){ //自执行  
 var b=0;  
 }();  
//4 function fn(){  
 console.log(num);//undefined  
 return function {  
 console.log(num);  
 }  
 alert();  
 var num=9;  
 }  
 var f=fn(); //函数的执行 赋值是执行的结果 没有return就是undefined  
 fn();  
//5 foo(); //2 预解释先赋值  
 function foo(){  
 console.log(1);  
 }  
 foo(); //2  
 var foo='haha'; //原来的函数引用地址已经被破坏，并且foo赋值了一个字符串，让一个字符串去执行  
 foo(); //foo is not a function  
 function foo(){  
 console.log(2); //预解释的时候还会赋值  
 }  
 foo();  
  
  
  
**作用域的销毁和内存释放**  
1如果**让一个堆内存释放**掉那么**就让这个堆内存没有被其他变量引用**，如果**函数在执行后**，**函数体内的某一部分被函数体外占用**，那么我们**运行后返回的堆内存**和**运行时产生的作用域（栈内存）都不会被释放**，如果没有占用，浏览器会主动回收内存空间  
2**函数运行的时候**，在**函数体内**定义的**匿名函数赋值**给函数**体外的dom元素的事件属性**，**内存不会释放**3**函数运行之后留下来的返回值是一个函数**，**立刻有执行一次**，**这样内存不会被立刻释放掉**。  
 var obj={name:"haha",age:20}; //这个对象的内存地址只要被obj这个变量引用，这个堆内存就不会被释放。  
 obj=null; //把一个空的指针赋值给obj，这样以前的内存就会被释放  
//1 var num=12;  
 function fn(){ //fn在预解释的时候已经固执一个堆内存  
 var num=100;  
 return function { //仍然是一个引用数据类型，堆内存地址xxxfff111  
 console.log(num);  
 }  
 }  
 var f=fn(); //是把fn运行留下来的结果赋值给f(并且我们发现return出来是一个函数,相当于把函数的引用地址赋值给f变量)，函数运行： 1 形参赋值 2 预解释 3 代码执行 没有return就是undefined  
 //函数运行的时候开辟了一个新的私有作用域  
 fn(); //xxxfff111运行 在return后面函数定义的  
//2 (function(j){  
 oDiv['onclick'] = function (){  
 console.log('点我a');  
 //  
 //oli[j]  
 //this  
 })(i);  
//3 function text(){  
 return function () {  
 console.log(1);  
 }  
 }  
 text();