**1. 变量声明提升**

\* 通过var定义(声明)的变量, 在定义语句之前就可以访问到

\* 值: undefined

**2. 函数声明提升**

\* 通过function声明的函数, 在之前就可以直接调用，相当于剪切过去，位置在变量提升的后面

\* 值: 函数定义(对象)

**3. 问题: 变量提升和函数提升是如何产生的?**

**4. 代码分类(位置)**

\* 全局代码

\* 函数(局部)代码

**5. 全局执行上下文**

\* 在执行全局代码前将window确定为全局执行上下文

\* 对全局数据进行预处理

\* var定义的全局变量==>undefined, 添加为window的属性

\* function声明的全局函数==>赋值(fun), 添加为window的方法

\* this==>赋值(window)

\* 开始执行全局代码

**6. 函数执行上下文**

\* 在调用函数, 准备执行函数体之前, 创建对应的函数执行上下文对象(虚拟的, 存在于栈中)

\* 对局部数据进行预处理

\* 形参变量==>赋值(实参)==>添加为执行上下文的属性

\* arguments==>赋值(实参列表), 添加为执行上下文的属性

\* var定义的局部变量==>undefined, 添加为执行上下文的属性

\* function声明的函数 ==>赋值(fun), 添加为执行上下文的方法

\* this==>赋值(调用函数的对象)

\* 开始执行函数体代码

**7.执行上下文栈**

在全局代码执行前, JS引擎就会创建一个栈来存储管理所有的执行上下文对象

在全局执行上下文(window)确定后, 将其添加到栈中(压栈)

在函数执行上下文创建后, 将其添加到栈中(压栈)

在当前函数执行完后,将栈顶的对象移除(出栈)

当所有的代码执行完后, 栈中只剩下window

**面试题**

if (!(b in window)) {

var b = 1

}

console.log(b) // undefined

这道题可以看做在if语句执行前 就执行了var b的语句

所以b in window为true if的判断条件为false

所以不会给b赋值 最后的结果为undefined





