正课:

1. \*\*\*\*Function

\*\*\*\*作用域链

\*\*\*\*\*闭包

2. \*\*\*面向对象OOP

封装，继承，多态

1. \*\*\*\*Function

作用域(scope)和作用域链:

什么是: 用途: 一个变量的可用范围

本质: 其实是一个保存变量的对象

为什么: 避免不同范围的变量间互相干扰

包括: 2级:

1. 全局作用域: window

保存全局变量: 优: 可重复使用，随处可用

缺: 全局污染

2. 函数作用域: 临时创建的活动对象AO

保存局部变量: 优: 不污染全局

缺: 不可重用，仅函数内可用

强调: 局部变量包含2中:

1. 在函数内var的

2. 参数变量也是局部变量

原理: 四个阶段:

1. 程序开始执行前:

在内存中创建执行环境栈(数组):

执行环境栈: 依次保存正在调用的函数的数组

函数调用后，会出栈

在栈中首先添加浏览器主函数的调用

创建全局作用域对象window

主函数引用全局作用域对象window

2. 定义函数时:

在window中用函数名创建变量

在window外创建函数对象保存函数的内容

函数名变量通过地址引用函数对象

函数对象用隐藏的scope属性，引用回自己诞生的作用域对象。——万一运行时缺东西，可以找诞生的作用域要！

3. 函数调用时:

在执行环境栈中添加当前函数调用

为本次函数调用创建活动对象AO:

活动对象: 专门保存本次函数调用所需的局部变量的对象——就是函数作用域对象

在活动对象中保存函数的局部变量

活动对象通过隐藏的parent属性引用window

变量的使用顺序: 先用AO中的局部变量，局部没有，才去window找。

4. 函数调用后:

函数的调用从ECS中出栈

导致: 函数作用域对象AO释放

导致: 局部变量一同释放！

作用域链: 由多级作用域对象，逐级引用形成的链式结构

作用: 存储所有变量(局部变量和全局变量)

控制着变量的使用顺序，先局部再全局

\*\*\*\*\*闭包:

什么是: 即重用一个变量，又保护变量不被篡改的一种机制

为什么:

全局变量: 优: 可重用

缺: 造成全局污染，易被篡改

局部变量:

优: 仅函数内可用，不会污染全局，也不会被篡改

缺: 不可重用

何时: 只要希望重用一个变量，又保护变量不被污染时

如何: 3步: 鉴别是否形成闭包的特征

1. 用外层函数包裹，要保护的变量和使用变量的内层函数

2. 外层函数返回内层函数

3. 调用者，调用外层函数，获得内层函数对象

缺: 闭包比普通函数占用更多内存！

解决: 用完后立刻释放！

getNum=null 导致: 内层小函数对象释放

导致: outer的AO释放

鄙视:

1. 先找受保护的变量和内层函数，画简图:

2. 同一次外层函数调用，返回的多个内层函数，共享同一个闭包变量

3. 多次外层函数调用，会创建多个闭包。多个闭包之间彼此毫无关系！

2. \*\*\*面向对象OOP

什么是对象: 描述现实中一个事物的属性和功能的程序结构

同时存储一个事物的多个属性和功能的存储空间

什么是面向对象: 程序中，都是用对象来描述现实中一个具体事物

为什么: 现实中每个属性和功能，必须依附在一个具体对象上才有意义！

何时: 只要管理多个事物,都必须用对象管理！

如何: 三大特点: 封装，继承，多态

封装:

什么是: 将一个事物的属性和功能集中定义在一个对象中

为什么: 1. 符合现实的习惯

2. 便于维护每个对象各自的属性和功能

何时: 只要使用面向对象的思想，第一步，都是封装对象

如何: 如何创建对象: 3种

1. 用{}直接量:

var obj={

属性名:值,

属性名:值,

... : ...

方法名(){//ES6

... ...

}

}

问题: 对象自己的方法如何动态访问自己的属性值？

解决:

错误1: 直接使用属性名!

报错: 找不到属性名变量

原因: 所有不加点的变量，只能在作用域链中找。无法自动进入对象中。

错误2: 用对象名.属性名

问题: 不便于维护! 对象名仅是普通变量，随时可能改名。

正确: \*\*\*\* this.属性名

什么是this: AO中自动定义的一个关键词

自动指向正在调用函数的.前的对象

为什么: 不受对象名的影响

何时: 只要对象自己的方法，要访问对象自己的属性，必须用this.属性名

补充: 事件处理函数中，也可用this自动获得触发事件的.前的元素对象本身。

访问对象中的成员: 属性和方法

属性: 其实就是保存在对象中的普通变量

比如: obj.属性名

方法: 其实就是保存在对象中的普通函数

比如: obj.方法名()

2. 用new创建: 2步:

var obj=new Object();//{}

obj.属性名=值;

obj.方法名=function(){ ... this ... };

揭示本质: js中一切对象底层，其实都是关联数组

对象其实就是关联数组的简化用法！

问题: 如果反复创建多个相同结构的对象时，代码繁琐，不便于维护！

解决: 构造函数

3. 用构造函数反复创建多个相同结构的对象:

什么是构造函数:

1. 定义同一类型的所有对象的统一结构

2. 将对象构造为想要的结构！

何时: 只要反复创建同一类型的多个对象时，都用构造函数

如何: 2步:

1. 定义构造函数:

function 类型名(属性参数列表){

this.属性名=属性参数;

this.方法名=function(){

... this ...

}

}

2. 用构造函数反复创建多个对象:

var obj=new 类型名(属性值列表)

new: 4件事:

1. 创建新的空对象:

2. ?

3. 调用构造函数，向对象中添加新属性

new自动修改this为正在创建的新对象

4. 返回新对象地址，保存到变量中

问题: 每个对象都创建一个方法的副本，浪费内存！

解决: