正课：

1. Function

2. 面向对象

1. Function

什么是: 程序中封装一段可重复执行的代码段的程序结构，再起一个名字。

何时: 只要一段代码可能多次重复使用时，都要封装在函数里。

如何: 2步:

1. 创建函数：3种：

1. 声明方式:

function 函数名(形参列表){

函数体;

return 返回值;

}——这个整体称为一个函数声明

强调:

1. function 在程序中是一个动词，每执行一次function 就会重新创建一个新函数。

2. 形参:

什么是: 专门接收传入函数的数据的变量——函数内部专用的局部变量

为什么: 函数体中，有些数据是不能写死的，是需要根据外部情况，动态变化的。

何时: 只要函数中有依赖于外部动态变化的值，就都要用形参来获得。

3. 返回值:

什么是: 函数内部的执行结果，被抛出到函数外部，让外部可以继续使用函数的处理结果。

为什么: 因为函数外部的调用者需要获得函数的执行结果。

何时: 只要函数外部的调用者需要获得函数的执行结果，继续后续操作时，就要用返回值。

js最核心原理之一:

函数其实也是一个对象

函数名其实只是一个普通的变量

函数名变量中保存着函数对象的地址值，通过内存地址，引用着函数对象。

当调用时，js引擎先找到函数名变量，再通过函数名变量中的地址，找到函数对象，再执行函数对象中保存的函数体。

声明方式创建函数的问题: 会被声明提前hoist

什么是: 在程序开始执行前，程序先找到所有var声明的变量和function声明的函数，提前到当前作用域的顶部集中创建。赋值留在原地。再开始顺序执行剩下的程序。

声明提前是js广受诟病的缺陷:

解决:

2. 用赋值方式创建函数:

优点：函数定义不会被声明提前

var函数名=function (形参列表){

函数体;

return 返回值;

}

结果: 除了不会被声明提前之外，其余用法和普通函数定义完全一样！

揭示了一个js最核心的原理之一:

函数名其实就是一个普通的变量

函数体其实是一个对象

3. 用new来创建:

var 函数名=new Function("形参1", "形参2",...,"函数体")

2. 调用函数：

var 变量=函数名(实参值列表)

原理: 通过函数名找到当前函数，执行函数体中的代码，将实参值列表中的数据，对位赋值给形参变量在函数内部使用。如果有返回值，则通过等号赋值给左边的变量。

强调: 一个函数，如果只是定义，而没有加()调用，则不会执行函数的内容。即使函数内容写错！也不报错！

重载(overload):

什么是: 相同函数名，不同形参列表的多个函数，在调用时可自动根据传入实参列表的不同，自动选择匹配的函数版本调用。

为什么: 减少函数的个数，便于调用者调用和记忆

何时: 一件事，可能根据不同的实参值，执行不同的操作时。

如何:

其它语言:

1. 先定义多个相同函数名，不同形参列表的函数

2. 用统一的函数名，调用函数，传入实参列表。

结果: 其它语言可自动根据传入的实参值列表不同，自动匹配对应的函数，调用。

问题: js不支持多个同名函数同时存在！所有，js不支持其它语言那种重载的方式。

其实，js支持重载效果，但是需要变通实现。

如何:

1. 既然多个函数不让定义，就定义1个！这一个函数中要包含所有情况的定义。

2. 用arguments对象自动接住函数外部传入的实参值列表。并判断arguments中的实参个数或实参值

arguments: 每个函数中自带的，专门自动接收所有传入函数的实参值列表的，类数组对象。

每个函数中自带: 不用我们手动创建

接收所用传入实参值得列表——用途

类数组对象: 长得像数组的对象

vs 数组:

相同: 1. 下标, 2. length

不同: 不是同一种类型。不是一家人。类数组对象不能使用数组家的函数。

当函数执行时，即使不定义形参，也可以随意传入任意多的实参值。不会报错。因为内部都有arguments对象接住所有的实参值。

匿名函数:

什么是: 定义函数是不定义函数名的函数

如何:

1. 回调函数：

什么是: 自己定义的函数，自己不执行，而是交给别的函数去自动调用执行。什么时候执行，执行几次，与咱们无关。

为什么: 节约内存，用完一次后，自动释放！

比如:

arr.sort(function(a,b){return a-b})

setTimeout(function(){ ... }, ms)

str.replace(/正则/ig, function(kword){

return 新值

})

2. 自调:

什么是: 定义完函数，立刻调用自己

为什么: 防止全局污染

全局污染: 保存在全局的变量，可能被任何位置篡改。——将来禁止使用全局变量

何时: 今后所有js代码，都应该包裹在匿名函数自调中。而避免使用全局变量。造成全局污染

如何:

(function(){

函数体

})();调用

说明: 加上匿名函数自调的代码，和不加匿名函数自调的代码功能，完全一样。只不过避免了使用全局变量，将来造成全局污染。

强调: 匿名函数自调之后，必须分号结束

作用域(scope)和作用域链(scope chain):

什么是: 一个变量的可用范围

为什么: 避免不同范围的数据之间互相干扰！

包括: 只有2级:

1. 全局作用域:

什么是: 在程序任何位置都可以访问到的，其实就是window对象。

何时: 如果一个变量可能夸多个函数反复使用，都要定义在全局。但是，今后，仍然禁止使用全局变量。

如何: 凡是不属于任何函数或对象内的，独立的变量和函数，都是全局变量和全局函数。

2. 函数作用域:

什么是: 仅在当前函数内可用的范围

何时: 绝大多数情况下，数据都是仅在当前函数内使用就够了。所以，绝大多数情况下优先用局部变量。

如何: 2种:

1. 在函数内var出的变量和function创建的内层函数，都是仅局部使用的。出了函数不可用

2. 函数的形参变量，虽然没有var，但是也是在调用函数时临时创建的局部变量。仅函数内可用

原理: 其实函数作用域，也是一个临时对象

如果一个函数仅定义，不调用，是没有临时的函数作用域对象的。也没有局部变量。

当调用一个函数时，才临时创建当前函数的作用域对象。并在作用域对象中创建局部变量

调用过程中: 优先使用局部变量，局部没有才去全局找。

函数调用后：临时的函数作用域对象就没人要了。就自动释放了。同时，局部变量一同释放。

所以，局部变量，不可重用！

js中没有局部(块级)作用域:

js中不是所有{}都是作用域的

比如: for(){ ... }

if(){ ... } else { ... }

while(){ ... }

... ...

都不是作用域，这些{}内部的变量出了{}依然可用

比如:

for(var i=1, sum=0;i<=100;i++){

sum+=i;

}

console.log(sum)

再比如:

if(...){

var sql="aaa"

}else{

var sql="bbb"

}

pool.query(sql,...)

vs java是三级作用域:

多了局部(块级)作用域

在java中: for(){ ... }

if(){ ... }else{ ... }

这些大括号都是作用域

在java中，这些大括号里的变量出了{}，都不能用了。

比如：以上两种写法，在java中都是错误的！

应该改为:

var sum=0;

for(var i=1;i<=100;i++){

sum+=i;

}

console.log(sum)

再比如：

var sql;

if(...){

sql="aaa"

}else{

sql="bbb"

}

pool.query(sql,...)

作用域链:

什么是: 由多级作用域对象，逐级引用，形成的链式结构

为什么: 函数执行时，难免自己缺少一些变量，需要从全局的公共区域查找。

如何:

1. 每个函数其实都有一个"好友列表"，记录着自己将来可以找谁借钱！

2. 函数不调用时，好友列表中，只有window，所有函数都将window当做备胎

3. 当函数调用时，临时创建了自己的函数作用域。会将临时的函数作用域加入好友列表中，window之前。

4. 当函数中用到变量时，按"好友列表"的顺序，在作用域链中查找变量使用。使用顺序:

先在局部找。

局部没有，才去全局找。

闭包: closure

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，还是局部变量都有不可兼得的优缺点:

全局变量: 优: 可重用， 缺: 易被污染

局部变量: 优: 不会被污染， 缺: 不可重用

何时: 如果希望给一个函数保存一个专属的可重用的变量时，都要用闭包

如何:

1. 用外层函数包裹受保护的变量和内层函数对象

2. 外层函数将内层函数的对象返回到外层函数外部

3. 使用者需要调用外层函数，获得返回的内层函数，再起一个名字。

结果: 受保护的变量，不会被全局的语句修改，且被当前内层函数反复使用了。

原理: 看视频

鄙视: 闭包是如何形成的？

外层函数调用后，外层函数的作用域被内层函数引用着，无法释放！

缺点: 比普通的函数多占用一块内存空间（父母函数的局部变量）

解决: 一旦闭包不再使用，应该立刻释放！

将保存内层函数的变量赋值为null，就释放了内层函数对象，同时释放了父母的函数作用域。

比喻: 简单描述什么是闭包:

父母生了一个孩子

给孩子包了一个红包，红包装在孩子兜里，只能孩子用。

单词列表:

1. hoist: 起重机

2. overload 重载

3. scope 范围

4. chain 锁链

5. closure 闭包，假发套

作业:   
定义求和的函数add()，可计算任意多个数字的和

function add(){

}

console.log(

add(1,2,3),//6

add(1,2,3,4,5),//15

add(1,2,3,4,5,6,7)//28

)

//定义函数getNum()

要求: 定义函数getNum()，反复调用，可以反复生成一个连续递增且不重复的数字。

getNum()//1

getNum()//2

getNum()//3

getNum()//4

要求，无论何时调用，无论中间有任何代码，都不能影响取号的顺序。