高阶函数：操作函数地函数，可以接收一个或多个函数作为参数，并返回一个新函数。

阶乘函数可以将上次的计算结果缓存起来。在函数式编程当中，这种缓存技巧叫做‘记忆’。

函数声明完全可以提升，变量声明则是部分提升（赋值不可以提升）。

具名参数通过名称来做变量地映射。在函数定义中名称会与形参做关联，在函数调用中，名称与实参作官联。

只要参数地名称正确，则不需要关心参数名地顺序。

JS原本不支持具名参数。我们可以通过对象字面量来命名参数，并将对象作为一个实参来传入函数。

程序：静态性（词法性）（描述性的词法就像是静态声明，涉及到了程序的词法（单词、源码））、动态性（表示一个程序在运行期所表现出来的特性）

变量的作用域：变量可以在何处被访问到。词法作用域：JS中的变量都有词法作用域，一个程序的静态结构决定了变量的作用域（这个作用域并不会被函数从哪里调用等影响）。

嵌套作用域：如果在一个变量的直接作用域中嵌套有多个作用域，那么这个变量在所有的这些作用域中可以访问到。

覆盖：如果在一个作用域中声明了一个与外层作用域同名的变量，那么在这个内部作用域以及其内部的所有作用域中将访问不到外面的变量。

而内部的变量的变化并不影响外部的变量，当离开内部作用域，外部变量又可以被访问了。

JS的变量是函数级作用域：只有函数可以产生新的作用域。

函数声明可以被整体提前，变量只能声明提前，赋值不可以。

IIFE:立即执行、必须是一个表达式、最后有分号。

可以使用前缀运算符来强制执行一个表达式，例：！function(){}();void运算符也可以。好处：分号可以省略。

可以使用参数的方式在IIFE的内部定义变量：var x=23; (function(twice){console.log(twice);}(x\*2));

IIFE的使用场景：避免全局变量、隐藏全局作用域的变量、创建新的环境、避免共享、保持全局的数据对于构造器的数据相对独立、将全局数据附加到单例对象上、将全局数据附加到方法中。

全局作用域两大缺点：程序不健壮、重用性低；变量名字冲突。

模块系统解决了全局变量的问题。模块不会通过全局作用域提供接口，并且每一个模块都为它其中的模块全局变量提供了自己的作用域。

全局对象：JS一些不常用的特性可以通过一个对象来为全局的变量创建环境，这个对象就是全局对象。

全局对象可以用来创建、读取或是修改全局变量。全局对象有原型：getAllPropertyNames()函数来罗列全局对象的属性。

浏览器中全局对象是window,文档对象模型（DOM）的一部分。Node.js的全局对象是global,是Node.js的专属变量。

每一个模块都有它自己的作用域，在这个作用域中this指向了这个作用域的对象。因此，this和global在模块中是不同的。

window的使用场景：创建全局变量、内置插件、风格检查器、检查一个全局变量是否存在（不常用）、在全局作用域中创建内容。

环境：提供存储空间的数据结构，将变量名与变量的值做映射。环境被存储在堆中。

变量的两种传递方式：动态维度:调用函数；词法维度：与外部作用域进行关联。

[[Scope]]：JS内部属性、记录函数的作用域。

函数调用时，JS会为这个函数所在的新作用域创建一个环境。这个环境有一个外层域（outer），它通过[[Scope]]创建并指向了外部作用域的环境。

JS中一直存在一个环境链，以当前环境为起点，连接了一层外部环境。

每一个环境链最终会在全局环境（它是所有函数初始化调用的作用域）终结。全局环境的外部环境指向了null。

闭包：函数+函数创建时所创建的作用域。闭包会造成环境共用。