正课:

1. \*\*函数:

\*\*\*作用域

\*\*\*声明提前

\*\*\*按值传递

2. 全局函数:

3. 分支结构:

1. \*\*函数:

什么是: 程序中封装一项任务步骤清单的代码段，再起一个名字。

为什么: 重(chong)用

何时: 只要发现程序中一项任务可能反复使用，就都要先封装一个函数中，再反复使用函数。

如何: 2步:

1. 声明:

function 函数名(参数变量列表){

函数体;

return 返回值;

}

参数:

什么是: 接收传入函数中必要数据的变量

参数用法和普通变量完全一样，只不过不用var声明

何时: 如果一项任务必须某些数据才能正常执行时，就必须定义参数

作用:

1. 可用于提醒调用者，如何正确使用函数

2. 让函数变得更灵活！

如何: function 函数名(参数变量1,参数变量2,...)

返回值: 函数执行的结果

何时: 如果调用者需要获得函数的执行结果时，就用return返回结果。

return: 2个作用:

1. 退出函数执行——可单独使用

2. 退出函数时，顺便返回结果

如果函数没有返回任何结果，就返回undefined

2. 调用:

什么是: 让引擎按照函数的步骤清单执行一项任务

何时: 函数只有调用才执行，不调用不执行！

如何: var 返回值=函数名(参数值列表)

\*\*\*作用域(scope):

什么是: 变量的可用范围

为什么: 避免不同范围的变量之间互相污染

包括: 2个范围:

全局作用域: window

什么是: 程序中的任何位置都可以访问的范围

保存着全局变量:

什么是: 保存在全局作用域中，程序任何位置都可使用的变量。

特点: 随处可用, 可反复使用

何时: 只要希望随处可用的公共变量

如何: 只要不属于任何函数的变量，自动都是全局的。

函数作用域:

什么是: 仅函数内可用的范围

保存着局部变量:

什么是: 保存在函数作用域中，仅函数内可用的变量。

特点: 仅函数内可用，不可反复使用

何时: 如果只限于当前函数内使用，函数外不可用时。

如何: 2种:

1. 在函数内声明的变量

2. 函数的参数变量也是局部变量

变量使用顺序: 优先使用函数内的局部变量

局部没有，才去全局找

\*\*\*声明提前(hoist):

什么是: 在程序开始执行前，都会将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。再开始执行程序。

问题: 破坏了程序默认的执行顺序

解决:

1. 建议: 将所有的变量和函数声明都集中在当前作用域顶部。

2. ES6: let 代替var

let禁止在声明之前提前使用变量

强制将声明集中在当前作用域顶部创建

限制: 不能用let反复创建相同名称的变量

3. 函数: 第2种创建函数的方式：

var函数名=function (参数变量列表){... ...}

用以上方式创建的函数不会被声明提前

\*\*\*按值传递:

什么是: 两变量间赋值或将变量传入函数作为参数时，其实只是将原变量中的值复制一个副本给对方。

原始类型的值: 修改新变量，不影响原变量的值

2. 全局函数: 了解

什么是: ES标准中规定的，浏览器已经实现的，不需要任何前缀.就可直接调用的函数

比如: Number() String() Boolean()

isNaN() parseFloat() parseInt()

编码解码: 特指对地址栏中的url内容进行编码和解码

什么是编码: 将url中包含的多字节字符编码为单字节字符

为什么: url规定地址中不能包含多字节字符

如果包含，会乱码

何时: 只要url中可能包含多字节字符，都要将其编码为单字节字符

如何: var code=encodeURI(url)

解码: 将编码后的单字节字符串解码回多字节原文

何时: 只要收到的是编码后的字符串，都要先解码再使用

如何: var str=decodeURI(code)

问题: url中不但不允许多字节字符，而且还不允许出现保留了字符: 比如 : /

但是encodeURI无法对保留字编码

解决: 用encodeURIComponent代替encodeURI

用decodeURIComponent代替decodeURI

eval: 专门执行一个字符串格式的js语句

3. 分支结构:

什么是: 让程序根据不同的条件执行不同的操作

何时: 只要让程序根据不同的条件执行不同的操作

如何:

1. 一个条件，一件事，满足就执行，不满足就不执行:

如果操作简单: 短路: 条件&&(操作1,操作2,...)

如果操作复杂: if结构:

if(条件){

操作;

}

2. 一个条件，两件事，二选一执行:

如果操作复杂: if...else结构

if(条件){

满足条件才执行的操作

}else{

不满足条件时执行的操作

}

如果操作简单: 三目/三元/条件运算:

条件?操作1:操作2;

如果只是根据不同的条件，使用不同的值:

条件?值1:值2

如果满足条件使用值1,否则使用值2

3. 多个条件，多件事，多选一执行:

如果操作复杂: if...else if...结构

if(条件1){

操作1

}else if(条件2){

操作2

}else if(...){

...

}else{

默认操作

}

强调: 1. 如果程序进入后一个条件，则暗示前一个条件不满足

2. 最后一个else可省略,后果,如果一个条件都不满足，则什么也不做。

如果操作简单: 三目

条件1?操作1:

条件2?操作2:

... ? ...:

默认操作 ——不能省略

如果只是根据不同条件选择不同的值:

条件1?值1:

条件2?值2:

... ? ...:

默认值

如果所有条件都是等于比较: 还可用switch case结构

switch(表达式){//首先计算表达式的值

case 值1: //如果表达式的值\*全等\*于值1

操作1; //则执行操作1

break;

case 值2://否则如果表达式的值\*全等\*于值2

操作2; //则执行操作2

break;

case ... :

... ... ;

break;

default: ——可省略

默认操作;

}

问题: switch意为入口

只要匹配一个case，就进入执行，并触发之后所有case的操作。

解决: 打断每个case之间的联系: break;

break: 专门负责中止并退出结构

如何: 在每个case之间都要加break