正课:

1. \*\*\*函数

2. 全局函数

3. 分支结构

1. \*\*\*函数:

什么是: 内存中封装一项任务步骤清单的代码段，再起一个名字。

为什么: 代码重用

何时: 只要一段代码，可能被反复使用，都要先封装在函数中，再反复调用函数。

如何: 2步:

1. 声明: function 函数名(参数列表){

函数体;

return 返回值;

}

参数: 函数运行时，接收传入函数的数据的变量

只不过不用var创建

参数列表: 多个参数间用逗号分隔

何时: 当一项任务必须某些数据才能正常执行时

为什么: 参数可让函数变得更灵活

返回值: 函数执行的结果

何时: 如果函数的调用者需要获得函数执行结果

如何返回: return 返回值;

其实，return可单独使用: 退出函数

如果没有返回值的函数，其实也返回东西:

默认返回undefined

2. 调用: 让引擎按照函数的步骤清单，执行任务。

var 返回值=函数名(参数值列表)

参数值列表: 传入函数的执行时必须的数据列表

只要函数定义时规定了参数变量，调用时都要传入参数值。且顺序和个数保持一致。

每个参数值之间用逗号分隔

强调: 函数不调用不执行，只有调用才执行。

\*\*\*作用域scope: 一个变量的可用范围

包括: 2种:

1. 全局作用域: window

全局变量: 直接定义在全局中，不属于任何函数的变量

特点: 可反复使用，随处可用

何时: 如果一个变量需要反复使用，或跨函数随处可用时

2. 函数作用域: ?

局部变量: 2种:

1. 函数的参数变量

2. 在函数内var出的变量

特点: 仅在函数内可用，出了函数不能使用

不可重用

何时: 如果一个变量，仅希望在函数内有效时

总结: 优先定义并使用局部变量

尽量少使用全局变量——避免被污染

变量的使用顺序: 优先使用函数内的局部变量

只要局部声明了，就不用全局的

除非局部没有声明，才去全局找

\*\*\*声明提前(hoist):

什么是: 在开始执行程序前，引擎会将var声明的变量和function声明的函数，提前到\*当前作用域\*顶部集中优先创建。再开始执行程序。——赋值留在原地。

鄙视: 如果先使用，后声明，一定在考声明提前

就要将所有var和function提前到当前作用域的顶部，先创建，再判断程序的输出

解决:

变量:

传统: 强烈建议将所有的声明都集中在当前作用域顶部。

ES6: let代替var声明变量

let要求: 在let之前，不允许提前使用该变量

函数:

var 函数名=function(...){...}

本质: 函数名其实仅是一个普通的变量

函数其实是保存代码段的引用类型的对象

函数名通过函数对象的地址引用着函数对象

\*\*\*按值传递:

什么是: 两变量间赋值，或向函数中传入参数时，其实只是将原变量中的值复制一个副本给对方。

结果:

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，不需要任何对象就可直接调用的函数。

比如: String()/Number()/Boolean()

parseInt/parseFloat()

isNaN()

反例: alert()/prompt()...

编码解码:

什么是: 将url中多字节字符或保留字符编码为单字节

为什么: url中不允许出现多字节字符以及和保留字符冲突的字符

何时: 只要url中包含多字节字符以及和保留字冲突的字符时，都要先编码为单字节。

如何: 编码: var code=encodeURI(url)

解码: var text=decodeURI(url)

问题: 不能对保留字符编码解码,比如: ":" , "/"

解决: 编码: var code=encodeURIComponent(url);

解码: var text=decodeURIComponent(url);

eval函数: 执行一段字符串格式的js表达式

程序三大结构: 顺序，分支和循环

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，只有满足才做，不满足就不做

如果操作简单: 短路: 条件&&(操作)

如果操作复杂: if(条件){

操作

}

2. 一个条件，二件事，二选一执行:

如果仅两个值二选一:

(条件?值1:值2)

如果操作简单: 三目/三元/条件

条件?操作1:操作2

如果操作复杂:

if(条件){

操作1;

}else{

操作2;

}

3. 多个条件，多件事，多选一执行

如果多个值，多选一: 三目

条件1?值1:

条件2?值2:

...?... :

默认值 ——强调: 默认值不能省略！

如果操作简单: 三目

条件1?操作1:

条件2?操作2:

...?... :

默认操作 ——强调: 默认操作不能省略！

如果操作复杂: if...else if结构

if(条件1){

操作1 //如果当前条件满足，则不再向后执行

}else if(条件2){

//如果进入条件2，暗示条件1不满足

操作2

}else if(...){

...

}else{ —— 如果没有默认操作，则else省略

默认操作

}

简写: 如果if/else if/else下只有一句话，可省略{}

——强烈不建议

特殊: 如果所有条件都是等于比较，可简写为

switch(表达式){ //先计算表达式的值

//用表达式的值和每个case后的值做\*全等\*比较

case 值1: //如果表达式的值全等于case后的值

操作1; //就进入case下执行操作

case 值2:

操作2;

...

default: //如果所有case的值都不等于表达式的值

默认操作; //则执行默认操作

} ——强调: 最后一个default可省略

问题: 只要前一个case满足条件，则后续所有case都被执行。

原因: switch是入口

解决: break; 可终止当前结构继续向后执行

如何: 每个case之间，都要加break

100题: 2, 4, 5