正课:

1. 数据类型转换:

2. 运算符和表达式:

1. 数据类型转换:

什么是: 将不需要的数据类型，转化为需要的数据类型

js四大特点之一: 弱类型: 3大特点:

1. 声明变量时，不需要提前指定变量的数据类型

2. 同一个变量先后可保存不同类型的数据

3. 不同数据类型间可相互转化

何时: 只要给定的数据类型不是想要的

如何: 2种:

1. 隐式转换: 不需要程序员干预，由js自动完成的数据类型转换

何时: 只要给定的数据类型和js程序需要的类型不相符，js就会根据自己的需要，自动转化数据类型。

比如: 算数计算中:

默认: 一切都转为number类型，再做计算

为什么: 因为number类型最适合计算

bool: true->1 false->0

特殊: +运算中，只要碰到一个字符串，那么一切都转为字符串，+运算变为字符串拼接！

2. 强制转换: 程序员主动调用函数完成的数据类型转换

何时: 只要给定的数据类型不是想要的，且自动转换的结果也不是想要的，就要强制转换。

如何:

其他类型转number: 2种:

1. Number(x): 将x转为number类型

何时: 都是隐式转换，其实相当于自动调用Number(x)，很少主动使用。

问题: 只能转换纯数字组成的字符串和bool类型

如果转不了: 就返回NaN

NaN: 不是一个数字的任何值 Not a Number

NaN参与任何运算结果只能是NaN

2. parseFloat/parseInt(str):

将str转为number类型，自动去除末尾非数字字符

parseFloat可保留小数部分

parseInt 去掉小数部分

何时: 只要将字符串转number，首选parseFloat

如果确实需要去掉小数，才选parseInt

如果转不了，也返回NaN

强调: 参数应该是string类型

如果给定的值不是string类型，则先执行隐式转换，转为string类型，再转number

比如: parseFloat(true)

parseFloat("true")->NaN

其他类型转string: 2种:

1. x.toString() 将x转为字符串类型

x不能是null或undefined ——不是万能

2. String(x) 将x转为字符串类型——万能

其实，隐式转为string时，都是自动调用String

其他类型转bool类型: 1种

Boolean(x)

规则: 只有5个值会被转为false:

0,"",NaN,null,undefined

其余都转为true

其实, 隐式转为bool时，都是自动调用Boolean(x)

练习中:

1. 凡是从页面上获得的一切都是字符串类型

2. typeof 变量 可输出变量中值的数据类型名

2. 运算符和表达式:

程序: 人的想法在计算机中的执行

运算符: 程序中模拟人的想法的特殊符号

表达式: 由数据，变量和运算符组成的完成一项单一任务的语句。

算数运算: +,-,\*,/,%

%: 模运算/取余数: 被除数/除数,不要商，要除不尽的余数部分

何时: 1. 取余数

2. 判断能否整除

判断偶数: 能被2整除 n%2==0

判断奇数: 不能被2整除 n%2!=0

隐式转换:

默认: 都转number

特殊: +运算中,碰到字符串，都转字符串，+运算变为字符串拼接。

舍入误差: 计算机中，也有计算不尽的数值

解决:

1. 按指定位数四舍五入: n.toFixed(2)

今后几乎所有显示钱数的地方都要toFixed(2)

2. 存储: 保存很长位数的小数: 0.39999999

关系运算: 用两个值做比较，做判断

包括: > < >= <= == !=

返回值: bool

隐式转换:

默认: 都转number

特殊:

1. 两个值都是字符串, 则不再转数字，而是依次比较每个字符的unicode号

2. 判断NaN:

NaN和任何值做> < >= <= ==5种比较时，永远false

NaN和任何值做!=比较是，永远是true

问题:NaN==NaN =>false 用普通的==无法判断NaN

解决: isNaN(num) 专门代替==，用来判断num是不是NaN

何时: 只要判断是不是NaN，都用isNaN(num)

反用: 判断一个值是不是数字: !isNaN(num)

3. 区分null和undefined

问题：==比较时，会自动将undefined隐式转为null

null==undefined => null=null

解决: === 全等: 类型必须先相同，值再相等

其实===就是不带隐式转换的==

4. ?

逻辑运算: 将多个关系运算，综合起来得到最终结论

何时: 只要根据多个条件，综合得出最终结论时

包括: &&而且 ||或(要么) !不(没有)

条件1&&条件2...:

要求所有条件都为true，结果为true

只要一个条件为false，结果为false

条件1||条件2...:

只要一个条件为true，结果为true

除非所有条件都为false，结果才为false

!条件: 颠倒条件的判断结果

隐式转换: 默认将每个条件都转为bool类型

短路逻辑: 如果前一个条件已经可以得出最终结论，则后续条件不再执行。

&&: 只有前一个条件为false时，后续条件才不执行

||: 只要前一个条件为true时，后续条件不再执行

利用短路:

1. 简单分支结构: 1个条件，1件事，只有满足才做

如何: 条件&&(操作)

练习中: 运算符优先级: 优先级高的运算，先计算

改变优先级: 用()

2. 实现默认值/备用值:

如何: 值1||值2

如果值1可以被转为true，就选择值1使用

如果值1不能转为true，就选择值2使用

位运算: 了解

1. 左移/右移:

1<<3=8 1左移3位 相当于1\* 2的3次方

2<<3=2\* 2的3次方=16

8>>3=1 8右移3位 相当于8/ 2的3次方

2. 取整:

n^0 n|0 n>>>0 代替/简化Math.floor(n)

3. 交换两变量的值:

var a=3,b=5;

方法一: var t=b; b=a; a=t;

方法二: a+=b; b=a-b; a=a-b;

方法三: a^=b; b^=a; a^=b;

赋值运算:

扩展赋值运算: 对特殊赋值运算的简写:

a=a+b 将a的值+b的值，再存回a中

——将b的值累加到a上

可简写为a+=b;

a=a-b -> a-=b;

a=a\*b -> a\*=b;

a=a/b -> a/=b;

a=a%b -> a%=b;

更简化: 递增递减运算: 如果每次累加1/累减1

a+=1 -> a++

a-=1 -> a--

前++和后++:

1. 如果单独使用, 前后都一样

2. 如果参与到其他表达式中时:

相同: 变量中的值一定都会+1

不同: \*前\*++，返回+1后的\*新\*值

\*后\*++，返回未+1的\*旧\*值

鄙视:

var n=3;

console.log(n++ + ++n + n++);

// ? ? ?

//console.log(++n + n++ + ++n);

// ? ? ?

console.log(n);//6

提示: 所有表达式都是从左向右依次执行

如果前一个表达式修改了变量的值，则会影响后续表达式的执行。