正课:

1. \*数据类型:

string boolean undefined null

2. \*\*数据类型转换

3. \*\*运算符和表达式

1. \*数据类型:

string:

什么是: 专门保存一串文字的数据类型

何时: 如果一串文字只用于给人看，就要保存为字符串

比如: 姓名, 手机号, QQ号, 电子邮件, 网址...

如何: 程序中凡是包裹在一对儿引号中的字符，默认就是字符串类型

存储空间:

js中采用unicode编码保存每个字符:

unicode编码: 人为对全球主要语言中每个文字编一个号

为什么: 计算机只认数字(二进制),不认字符

比如: 张 24352 -> 二进制

所占内存空间: 数字"0"~"9" 48~57 1字节

字母"A"~"Z" 65~90 1字节

"a"~"z" 97~122 1字节

英文标点: 1字节

汉字"\u4e00"~"\u9fa5" 2字节

"Hello张东" 9字节

100000000 4字节

"100000000" 9字节

boolean:

什么是: 专门保存判断结果的类型

何时: 只要保存一个条件判断的结果，就用boolean

如何: 只有两个值: true false

强调: 不要加引号！

undefined 和 null

undefined 专门由程序自动给变量赋初值

表示一个变量仅声明，暂未赋值

null 专门让程序员手动清空一个变量之用

2. \*\*数据类型转换:

什么是: 将数据本身的类型，转化为你需要的另一种数据类型

何时: 只要给定的数据类型和运算要求的数据类型不相符，都要先转化数据类型，再执行运算

\*\*\*js语言是弱类型语言: 3大特点:

1. 声明变量时，无须提前指定变量的数据类型

2. 一个变量，先后可保存不同类型的数据

3. js程序会根据自身的需要，隐式转换数据的类型

包括: 2种情况:

1. 隐式转换: 无须程序员干预，程序自动完成的数据类型转换

何时: 只要给定的数据类型，不符合正在执行的js程序的要求，程序都会先转换为想要的数据类型，再执行程序。

比如: 仅以算数运算为例:

默认: 都转为number , 其中: bool: true->1 false->0

特殊: +运算中,只要碰到字符串，就都转为字符串，且加法计算变成字符串拼接！

2. 强制转换: 由程序员主动调用功能，实现的数据类型转换

何时: 只要隐式转换的结果还不是想要的!

包括:

转number: 2种:

凡是页面上获得的字符串，都要先转为数字，再执行算数计算.

1. 转换纯数字组成的字符串和bool为number

Number(x) 返回一个number

如果转不了, 返回NaN

NaN: Not a Number 代表一切不是数字的值

将来，只要获得NaN，说明无法转换

其实Number(x)很少主动使用，几乎都是隐式自动调用

2. 将带单位的字符串，去单位后，再转数字:

parseFloat(str) 保留小数部分 ——首选

parseInt(str) 去掉小数部分——除非确定去掉小数部分时

强调: Number(x)，什么都可接收

parseFloat/Int(str), 默认接收字符串

比如: Number(true) -> 1

parseFloat(true) -> NaN

parseFloat("true")

转string:

x.toString() 不能转null和undefined

String(x) 万能! ——首选

其实，隐式转字符串，都是自动调用String

专门的用途:

一般查看一个变量/数据结构的内容时，才转字符串

——拍照

转 boolean: 几乎不会主动使用!

Boolean(x)

\*\*\*规则: 只有5个值: "" 0 NaN null undefined 转为false，

其余都转为true！

3. \*\*运算符和表达式:

什么是程序: 人的想法在计算机中的执行

什么是运算符: 程序中模拟人的想法的特殊符号

什么是表达式: 由运算符，数据和变量组成的完成一项任务的一条程序语句。

所有表达式默认从左向右，两两执行

算数运算: + - \* / %

%: 模运算/取余数/取零头

m%n =? m/n 不取商，取除不尽的余数部分:

12345%100=45

20000%100=0

8%3=2

何时使用: 1. 取余数/取零头

2. 判断整除/奇偶数/闰年

8%2=0 8是偶数

7%2=1 7是奇数

隐式转换:

默认: 都转number，再算数计算

特殊: +运算中，只要碰到字符串，就都转字符串，+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

不可避免!

近似解决: 按指定小数位数四舍五入!

如何: n.toFixed(d) 将n按d位小数四舍五入

比如: n.toFixed(2) 将n按2位小数四舍五入

今后，凡是给客户显示的钱数，都要按2位小数四舍五入

如果算数计算中，任意一个值为NaN，则计算结果，也是NaN

关系运算: 做比较，做判断

返回值: bool类型的true/false

包括: > < >= <= == !=

隐式转换:

默认: 都转为number类型

特殊:

1. 两个字符串: 不再转数字，而是按位比较每个字符的unicode号, 直到比出大小为止.

2. null和undefined 无法用==判断

因为在底层，null和undefined是相等的.

解决: 用===全等 代替 ==

=== 不带隐式转换的==比较:

要求: 必须类型先相同，且值相等

请用户输入: var input=prompt("输入提示");

扩大: 只要希望两值不但值相等，且类型也必须相同时

就可用===代替==

3. NaN也不能用==判断:

NaN不大于，不小于，不等于任何值

解决: isNaN(num) 专门代替==，用于判断给定的值是不是NaN

反用: !isNaN(num) 专门判断num是不是数字！

不是NaN -> 就必然是数字！

强调: \*\*\*今后凡是从页面上获得数据，都是字符串类型！

逻辑运算: 将多个关系运算综合起来，得出最终的结论

返回值: bool类型: true/false

包括: 3种:

&&: 而且

条件1&&条件2: 必须同时满足条件1和条件2,才返回true，只要有一个条件为false，则结论为false

||: 或者/要么

条件1||条件2: 只要满足任意一个条件，就返回true。

除非所有条件都不满足，才返回false

!: 不

!条件: 颠倒条件的结论

隐式转换:

默认将每个条件都转为bool类型，再综合判断

Boolean(条件1)&&Boolean(条件2)

Boolean(条件1)||Boolean(条件2)

! Boolean(条件)

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:

&&: 如果前一个条件为true, 则后一个条件继续执行

如果前一个条件为false, 则后一个条件不再执行

||: 如果前一个条件为true, 则后一个条件不再执行

如果前一个条件为false, 则后一个条件继续执行

利用短路:

1. 利用&&的短路: 简单分支结构: 1个条件1件事，满足才执行，不满足就不执行！

条件&&(操作): 只有条件满足，才执行操作，否则不执行 操作

2. 利用||的短路: 默认值/备选值:

值1||值2: 如果值1有效(不是0,"",NaN,null,undefined)

就首先选择值1使用

如果值1无效,就选择备用值2使用

赋值运算: =

可简化: 扩展赋值运算:

何时: 只要先取出变量的值修改，之后再存回变量中

比如: total=total\*0.8

如何: a=a+b 可简写为 a+=b

a=a-b 可简写为 a-=b

a=a\*b 可简写为 a\*=b

a=a/b 可简写为 a/=b

a=a%b 可简写为 a%=b

更简化: 如果每次只是+1或-1

a+=1 可简写为 a++

a-=1 可简写为 a--

仅以++为例: 其实++可写前，也可写后:

++a vs a++

相同: a的值一定都被+1

不同: ++a 返回+1后的\*新\*值

a++ 返回+1前的\*旧\*值

位运算: 了解

1. 左移和右移:

左移: m<<n 等效于 m\* 2的n次方

m>>n 等效于 m/ 2的n次方

比如: 1<<3=8 2<<2=8

16>>2=4

2. 取整: 鄙视

m^0 m|0 m>>>0 用于简化parseInt(m)

3. 不声明第三个变量，交换两个变量中的值: 鄙视

a^=b; b^=a; a^=b;

作业:

var n=3;

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

//所有表达式，默认都是从左向右依次执行！

//先执行的表达式，会影响后续表达式中的变量

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

console.log(n);//6