语句，值和变量

语句和注释

在开始变量之前我们讲一下语句。JS 的语句是以英文分号 `;` 结尾的，而**不是换行**，因为同一行内可以用分号隔开多条语句。虽然 JS 对于一个没有分号直接换行的语句会自动补上分号，但是最好不要这么做。为了防止自己出现这些问题，你可以给自己的 VSCode 装上相关的 Lint 插件。

注释可以写在代码里面，解释器会自动忽略它们。你可以用这些说明你的程序干了什么。JS 里面有着两种注释，一种是用 `/* */` 包裹的注释，一种是用 `//` 开头到一行结尾的注释。下面是两种注释的使用。后面我们会用到的。

```
/* 我是一行注释 */
// 我也是一行注释

/*
我可以跨行
*/
// 我只能
// 这么跨行
```

数据类型与基本运算

JS 里有七种基本数据类型，但是很多类型现在都用不到。所以我们就先讲下最常用的数字和字符串，以及一些基础的运算。

数字

在 JS 里面只有一个数字类型。不论这些数字是像30（也叫整数）这样，或者像2.456这样的小数（也叫做浮点数），在内部都是用浮点数来表示的。这个和别的编程语言（比如 python）不同。这意味着 `1` 和 `1.0` 是同一个数字（而其他语言就是两个类型的数字）。

数字可以进行四则运算，优先级和数学里一样，你也可以加上括号改变优先级，就像在用计算器。乘号使用星号(`*`)代替，除号使用斜杠(`/`)代替。运算符两边可以加上空格，这样排版会更美观一点(也许是)。

你可以打开浏览器的控制台运行你的 JS，也可以在本地安装 Nodejs ，使用 Node 来运行。我演示的时候使用的是 Node，因为可以直接在 VSCode 的控制台内输入 `node` 指令来打开。

下面是在 Node 中运行的结果，其中 `>` 表示这行是输入，输入紧跟着的下一行就是控制台的提示输出。注意实际的 JS 脚本在浏览器运行的时候是没有给每行加上一个输出的，你要用 `console.log()` 来进行。

```
> 1; // 1 就是 1
1
> -1; // -1 就是 -1
-1
> 1+1;  // 就是一加一
2
> 1 + 1; // 加上空格之后运行结果也一样
2
> 1*2 + 2*3; // 运算符优先级和数学一样
8
> (1+2)*3; // 使用了括号
9
> 6 / 3; // 除法
2
> 1.1 + 1.5; // 小数加法
2.6
> 0.1 + 0.2; // 这是浮点数误差导致的
0.30000000000000004
>
```

最后一个 `0.1 + 0.2` 算出来结果是 `0.30000000000000004` （15 个 0 ，不用数了），这是怎么回事？这不是 bug，是语言的特性，叫做**浮点数误差**，是因为计算机在转换小数到二进制的时候丢失精度导致的(可以看 https://0.30000000000000004.com/ 了解)。所以以后判断小数相等的时候可能不能直接用等号来进行。

除了四则运算之外，还有取余运算和乘方运算，取余运算使用百分号(`%`)，乘方运算使用两个乘号(`**`)。在两者都是正数的时候，取余可以取得前一个数字除以后一个数字的余数。如果出现负数可能就不太一样，不过一般不会遇到，所以就先略过。

```
> 5 % 2;
1
> 5 % 3;
2
> 25 % 5;
0
> 2**15;
32768
> 2**3;
8
>
```

有时候我们需要舍弃小数部分，这可以用 `Math` 的一些方法来进行。其中 `Math.round()` 四舍五入，`Math.floor()` 向下取整， `Math.ceil()` 向上取整。

```
> Math.round("3.1415926")
3
> Math.round("3.999")
4
> Math.ceil("3.1415926")
```

```
4
> Math.floor("3.999")
3
>
```

数字的范围是有限制的，超出限制的就无法精确表示。JS 能够准确表示的整数范围在 `-2^53` 到 `2^53` 之间（不含两个端点），超过这个范围，就无法精确表示这个整数。

如果一个式子不能被计算，那么它的值就会变成 `NaN`（Not a Number，不是一个数字，最典型的就是给 -1 开平方，或者尝试给一个不全是数字的字符串转成数字。

```
> Math.sqrt(-1); // 给 -1 开平方
NaN
> parseInt("123"); // 正常的转换
123
> parseInt("Your life is which you loved"); //给字符转数字
NaN
>
```

字符串

另一个非常常用的就是字符串。顾名思义，这就是一串的字符。它可以用一对英文单引号（''）或者一对英文双引号("")包裹一串字符来表示，比如 `'我是一个字符串'` `"我是另一个字符串"`。

同样和别的语言不一样的是，JS 只有字符串类型，而没有单个字符类型。你可以用只有一个字符的字符串代替。

如果要在字符串里面包含特殊字符可以在字符前加上反斜杠(`\`)进行转义。比方说要包含引号的时候，就可以用 `\"` 或者 `\'` 来表示引号本身而不是字符串结尾。

你可以使用加号 `+` 来对字符串进行拼接操作，这可以得到这些字符串首尾相连的结果。

```
> "I am a str"; // 使用一对双引号
'I am a str'
> 'Another str'; // 一对单引号
'Another str'
> 'a'; // 单个字符的字符串
'a'
> '一个有引号\"的字符串'; // 转义
'一个有引号"的字符串'
> '也可以这样带上单引号:\'. '; // 也是转义
"也可以这样带上单引号:'. "
> "aa" + "bb"; // 拼接
'aabb'
> 'abc' + ' ' + 'def'; // 多个一起拼接
'abc def'
>
```

转换

字符串和数字之间可以相互转换。这些转换通过一些 JS 内置的函数进行。有些是显式进行的（比如调用一个函数），有些是隐式进行的（比如让字符串和数字相加）。

一般情况下，涉及数字转字符串的我们采取隐式的方式。字符串转数字的时候我们使用 `parseFloat()` 和 `parseInt()` 函数来显式转换，用法是在括号内放一个字符串。`parseFloat()` 的结果如果有小数会带上小数部分，`parseInt()` 的结果会直接去掉小数部分（而不是四舍五入）有关函数更为具体的内容我们以后会探讨，这里可以简单看作是一些封装起来的代码，执行之后返回一个结果。

下面我们来演示这些。

```
> "The answer is " + 42; // 一个拼接
'The answer is 42'
> 42 + " is the answer"; // 另一个拼接
'42 is the answer'
> "37" + 7; // 加法会把数字转成字符串
'377'
> "37" - 7; // 除了加法都会将字符串转成数字
30
> parseFloat("37") - 7; // parseFloat 的结果也一样
30
> parseFloat("123"); // 这里也是整数
123
> parseFloat("123.45"); // 小数部分也正常
123.45
> parseFloat("10e5"); // 还可以用科学计数法
1000000
> parseInt("123"); // parseInt 的整数转换
123
> parseInt("123.45"); // 现在去掉小数了
123
>
```

变量

变量本质上是值（例如数字或字符串）的容器，比如你可以用它去存一个数字 `123`，也可以存一个 `Hello world!` 字符串。

变量可以通过这三个关键字声明：

- `let`: 这是在 ECMAScript 2015 里面添加的。我们建议如果没有支持旧浏览器的需求就使用它。
- `var`: 这是旧的声明变量方式，现在也可以用，但是不推荐，因为 `var` 可以被重复声明，这会导致很多问题。
- `const`: 这是常量，声明之后不能修改。

> 最好不要使用 var，因为它可以被重复声明，如果不小心重复声明了一个变量，会导致问题问题。同时用 var 声明的变量会发生变量提升，产生更多难以预料的问题。

变量名在 JS 里面又叫做**标识符**。标识符是有规则的，必须以字母、下划线（_）或者美元符号（$）开头，而不能是数字开头；但是后续的字符也可以是数字（0-9）。同时 JS 支持 Unicode 字符集（也就是你可以声明一个中文变量，虽然没啥用）。JS 是大小写敏感的，比如 `aA` 和 `aa` 就是两个变量。

除了上面的规则之外，你也不能使用 JS 的保留字给变量命名。保留字，即是组成JavaScript 的实际语法的单词。比方说什么 `if`, `while`, `for` 这些。一般变量命名是不会和保留字重复的。

我们建议使用大小写驼峰命名法，也就是包含多个单词的变量名称的第一个单词开头小写，其余单词开头大写。比如我要命名一个 "My first name" 来存名字，变量名就可以写成 `myFirstName` 。

变量声明之后，默认的值是 `undefined` 。你可以通过比较一个变量是否等于 `undefined` 来判断变量是否初始化(使用双等号 `==` 或三等号 `===` 来比较，例如 `x === undefined;`)。直接在控制台里面打一个变量名可以输出变量的值，如果调用没有声明的变量会弹出一个错误。

非 `const` 变量在声明之后可以重新赋值。赋值使用单个等号，格式为 `变量名 = 新的值;` 。新的值可以是单个值，比如 `x = 1;` 也可以是一个表达式 `x = 1 + 2;`, 或者 `x = x + 1;` （给 x 加上 1）。

声明和赋值可以写在一起，比如 `let x = 1;` 就可以声明一个变量，值为 `1`。

我们现在可以这样试着声明变量，下面是在 Nodejs 上运行的结果，当然浏览器也一样。(语句**一定**要加上分号，虽然没有分号也能运行，但是最好不要养成坏习惯):

```
> let myFirstName; /*声明了一个 myFirstName*/
undefined
> myFirstName; // 显示 myFirstName 的值
undefined
> var myValue; // 声明了一个 myValue
undefined
> myFirstName = "Steve"; /*给 myFirstName 赋值了 "Steve" */
'Steve'
> myFirstName;
'Steve'
> myValue = 1 + 1; /*给 myValue 赋值了 1 + 1，值是 2 */
2
> myValue;
2
> myValue++; //这是自增运算，就是给 myValue 的值加上 1，循环的时候经常用
2
> myValue; // 现在 myValue 变成了 3
3
> myValue = "aaa"; /*重新给 myValue 赋值了一个字符串 */
'aaa'
> myValue; /*现在 myValue 是一个字符串*/
'aaa'
> let myFirstName;// 这里重复声明没有通过，而是报错了
Uncaught SyntaxError: Identifier 'myFirstName' has already been declared
> var myValue; /*这里重复声明，而且通过了*/
undefined
> myFirstName;
'Steve'
> myValue; /*还保留原来的值*/
1
```

```
> const myConst; /*声明 const 一定要先初始化！*/
const myConst;
      ^^^^^^^

Uncaught SyntaxError: Missing initializer in const declaration
> const myConst = 2; /*声明了一个 const 变量*/
undefined
> myConst; /*打印 myConst 的值*/
2
> myConst = 3; /*尝试给 myConst 重新赋值，不被允许*/
Uncaught TypeError: Assignment to constant variable.
>
```

你也可以现在就打开浏览器的控制台输入这些，来进行尝试。上机实践对学习编程非常重要，不上机很多问题在纸上看看是没有结果的。

由于 JS 的一些奇怪问题，我们建议不能使用下面的方式：

```
> bad1 = "This is bad"; /*没有加上关键字，自动变成全局变量 */
'This is bad'
> bad1;
'This is bad'
> bad2 === undefined; /*这是一个没有声明的变量的正常行为*/
Uncaught ReferenceError: bad2 is not defined
> bad2 === undefined; var bad2 = 0; /* 先使用后声明，发生变量提升*/
true
> bad2;
0
> bad3 = 0; let bad3; /*所以要用 let*/
Uncaught ReferenceError: Cannot access 'bad3' before initialization
>
```

用 `var` 声明的变量会发生变量提升，相当于把声明语句往上挪到作用域的开头了。这没什么用，反而会造成一些问题，所以你应该尽量只用 `let` 来声明。

变量赋值之后就可以用来进行一系列运算，就和普通的值一样。

```
> let x = 2; // 声明+初始化
undefined
> let y = 3;
undefined
> x + y; // 进行四则运算
5
> x - y;
-1
> x + 10;
12
> let str1 = 'AAA'; // 声明了两个字符串
undefined
```

```
> let str2 = 'BBB';
undefined
> str1 + str2; // 尝试一些拼接操作
'AAABBB'
> x + ' ' + str1;
'2 AAA'
> str1 + " is Str1";
'AAA is Str1'
>
```

输入和输出

不像其他语言，JS 最初是给浏览器设计的，直到现在这也是它的最主要用途，所以它和 DOM 深度绑定，信息几乎都是从 HTML 页面读取，输出也基本都是写在HTML 当中，甚至没有标准化的输入和输出。由于我们现在还没学过对象(Object)，所以我们采用一些其他方式进行输入输出。

我们使用 `console.log()` 函数进行输出，这个函数会在控制台上留下一条信息。不仅可以给它一个字符串，也可以给它一个语句，比如 `console.log(1 + 1);` 输出 2。

对于输入，我们选择手动更改代码中的相关变量赋值来进行替代。这可能有点麻烦，不过在学了 DOM 之后就可以用更标准的方法进行。

最后

我们了解了 JS 里面的数字和字符串类型，以及它们的基本运算和转换关系。

我们还了解了变量，它是一种容器，可以用来存放变量。变量在赋值之后可以进行一系列的操作，就像普通的值一样。

现在，你可以写一些小程序来进行一些简单的运算了。比如一个最简单的 A+B Problem，输出 a+b 的结果:

```
let a = 1;
let b = 1;
let c = a + b;
console.log(c);
```

试试写出类似的 A-B，A*B，然后按照我们之前 Hello world 的方式加到浏览器里面运行。

这里的程序就是一个典型的**顺序结构**，代码从上到下一步步运行，最后得出结果。这是程序的三大基本结构之一，另外两个分别是**分支**和**循环**，我们会在不久之后提到。了解三大基本结构之后，理论上你就可以写出一些真正能用的程序了。

练习

写结果

写出以下代码的运行结果，一行一个。

例:

```
console.log(1 + 1);
console.log(2 - 1);
```

输出：

```
2
1
```

问题：

```
console.log((1 + 2) * 3);
console.log(2**3);
console.log((2 + 5) % 5);
```

```
console.log("Roses" + " are red.");
console.log("The number is " + 443);
console.log("123" + 23);
console.log("123" - 23);
```

```
let x = 12;
console.log(x);
x = x + 1;
console.log(x);
let y = x / 2;
console.log(x);
console.log(y);
```

```
let x = "Nether"
let y = "Mobs"
let z = x + y;
console.log(z);
console.log(y + 12);
```

最好自己试着写出来，然后用我们之前在 Hello world 里面提到的运行方法加到浏览器里面进行运行进行对比。

编程练习

有人说 Tux 太胖了， Tux 很生气，于是 Tux 想拿出数据来反驳。现在 Tux 要你写一个程序来计算 BMI，并在控制台输出。

下面是程序的开头，你要用给出的体重 (变量 `weight`) 和身高 (变量 `height`) 来计算，并存在变量 `bmi` 内，最后用 `console.log()` 输出。（BMI=体重/身高^2）

```
const weight = 82;
const height = 1.61;
let bmi;
```

李华在翻译一篇英语文章的时候，因为键盘上的反斜杠和竖杠的键坏掉了，打不出 "·" 号，所以就没法翻译外国人的名字了。现在他要你写一个程序，给名字的姓和名之间加上一个点，好让他完成作业。

下面是程序的开头，给出了姓 (`lastName`) 和名 (`firstName`)，在控制台输出在姓名中间加上点的结果。（不要忘记外国人名字和姓的顺序是反着来的）

```
const lastName = '莫伯斯';
const firstName = '尼泽';
```