## for 语句

while 和 do-while 语句可以用于实现循环结构，除此以外，JavaScript 中提供了另一种更加灵活快捷的方式来进行循环（或者叫迭代）：*for 语句*。

一个 for 语句看起来像这样：

for (初始化表达式; 条件; 增量表达式) {  
 执行语句  
}

等等，for 语句的小括号中包含了三个东西！它们都是什么？

首先是*初始化表达式*。它用于说明哪些值会被用在循环中。例如，在上一节的阶乘示例中，我们在进行计算之前要先设置变量 i 和 n 的值为 1，这类操作就是*初始化*。你可以将初始化的操作直接放在 for 语句里，称为初始化表达式。

初始化表达式的分号后面是*条件*，它表示控制循环进行的条件，与 while 语句是一致的。

而*增量表达式*用于在每次循环后，改变控制循环的变量的值，以此达到控制循环次数的作用。

事不宜迟，用一个简单的示例来看一下 for 语句的使用：

for (let i = 0; i < 5; i += 1) {  
 alert(i);  
}  
// 0  
// 1  
// 2  
// 3  
// 4

1. 首先设置控制循环的变量 i 的值为 0。
2. 查看 i 的值是否满足条件。
3. 满足，那么执行循环体的语句，显示出 i 的值。
4. 循环体结束，根据增量表达式，改变 i 的值，为下一次循环做准备。
5. 回到步骤 2。

如果 i 的值一开始就不满足条件，那么循环体一次也不会被执行，像 while 语句一样。

上一节中的计算 1 + 2 + 3 + ... + 100 的示例，使用 for 语句可以改写如下：

let n = 0;  
for (let i = 1; i < 100; i += 1) {  
 n += i;  
}  
alert(n); // 5050

我们使用 i 作为控制循环的变量，每次循环后它的值便会 +1 ，同时 i 也起到了从 1 增长到 100 ，用于使 n 进行累加的作用。

事实上，为了充分利用 for 语句，我们还可以更进一步：

for (let i = 1, n = 0; i < 100; i += 1, n += i) {  
  
}  
alert(n); // 5050

我们可以在初始化表达式的位置上写几个用于进行初始化的表达式，只需使用逗号隔开。

同样，增量表达式也可以对不同的变量进行增量处理。它的求值顺序是从左到右的，也就是说，先计算了 i += 1 ，然后再处理 n += 1。

如果我们的循环体没有语句，我们根本就不用写大括号，直接省略就好了！。

for (let i = 1, n = 0; i < 100; i += 1, n += i)  
  
alert(n);

但是运行这段代码，就会发现，alert 不会在循环结束后执行，而是每进行一次循环都会显示一次当前 n 的值。因此，你一共要点一百次“确认”，直到循环结束。

现在，刷新页面，一切恢复正常。

为什么会这样？因为 for 语句和 while 语句一样，如果没有写大括号，会将循环头部（即一对小括号包裹的内容）的后面遇到的第一个语句当做是循环体，因此 alert(n) 被循环执行了。解决这个问题的办法是在循环头部后面写一个分号 ; ，用一个*空语句*来代替循环体。

for (let i = 1, n = 0; i < 100; i += 1, n += i) ;  
// ; （更推荐写在第二行）  
alert(n); // 5050

练习 4.4.1

1. 创建一个 1 ~ 100 的循环，当数字是奇数时，打印“奇数”，否则打印“偶数”。

* 提示：使用运行器提供的 print 函数来进行“打印”操作。

1. 显示一个九九乘法表，使用 print 函数来打印每一行。

* 提示：你需要将一个 for 语句写在另一个内部，使用它们的控制变量来输出因数。

### 迭代算法

我们在这里第一次接触*算法*一词。Wikipedia 对此的定义如下：

在数学和计算机科学中，**算法**是一个明确的、关于如何解决一类问题的规范。算法可以执行计算，数据处理、自动推理和其他任务。算法可以在有限的空间和时间内表达。从初始状态和初始输入开始，描述了一系列计算，当执行时，通过有限个明确定义的连续状态，最终产生“输出” ，并终止于最终结束状态。

当我们要实现某种功能时，我们将会用精确的语言，描述我们所要实施的步骤。例如在上文中对于循环过程的描述就叫算法。迭代算法则是用迭代等方式实现的算法，通俗来讲，使用循环和条件控制来进行一系列运算，以得到我们需要的结果。上文中的累加和阶乘等运算就属于迭代算法。

我们将会在本节深入了解迭代算法，一个常见的实际应用就是检测一个数是否为质数。

如果你忘记了什么叫质数，我们可以先复习一下小学数学书上对于质数的描述：

如果一个大于 1 的整数只有 1 和它本身两个因数，那么它就是一个质数，否则就是合数。

2 是最小的质数，1 既不是质数也不是合数。

换句话说，一个整数如果大于 1，并且不能整除除 1 以外的所有比它小的整数，那么它就是一个质数。

我们很快就可以得到判断一个数 n 是否为质数的思路：

1. 如果这个数不大于 1 或不为整数，那么它必定不是质数。
2. 从 2 开始，列举从 2 到 n-1 的所有整数，用 n 除以列举的数。
3. 如果得到的余数为 0，说明 n 可以整除它，那么 n 不是质数。
4. 如果列举完后也没有找到一个数可以被 n 整除，那么 n 是质数。

我们可以根据这个思路尝试写出代码，列举数字的工作自然就交给 for 语句。

let n = 100;  
let isPrime = true; // 假设它是一个质数  
for (let i = 2; i < n; i += 1) {  
 if (n % i === 0) {  
 isPrime = false;  
 }  
}  
alert(isPrime ? "质数" : "合数"); // "合数"

通过列举它可能的因数，尝试进行整除，这种策略称为*试除法*。

这个代码可以正常工作，但我们很快就发现了它的问题：当我们发现 n 不是一个质数的时候，应该终止计算并告知结果。但是这里，即使我们发现了 n 是合数，for 语句也不会停下来，又白白将剩下的循环运行完。

因此，我们应当采取策略：当我们知道它不是一个质数的时候，我们就不再进行试除了，而是报告结果。

let n = 100;  
let isPrime = true; // 假设它是一个质数  
for (let i = 2; i < n; i += 1) {  
 if (n % i === 0) {  
 isPrime = false;  
 break;  
 }  
}  
alert(isPrime ? "质数" : "合数"); // "合数"

这里再次出现了 break; 语句。它在这里的作用是**直接终止循环**。

我们知道，一个合数最大的因数不会超过它的平方根，例如 100 最大的因数就是 10 ，判断一个整数是否为质数，只需列举到它的平方根进行试除就足够了：

let n = 100;  
let isPrime = true; // 假设它是一个质数  
for (let i = 2, last = Math.sqrt(n); i <= last; i += 1) {  
 if (n % i === 0) {  
 isPrime = false;  
 break;  
 }  
}  
alert(isPrime ? "质数" : "合数"); // "合数"

由于除了 2 以外的所有质数都是奇数，因此我们可以进行一个简单的判断：

* 如果输入的数字是 2 ，那么它是一个质数。
* 如果输入的数字不是 2，但能被 2 整除，那么它是一个合数。
* 从 3 开始列举它的因数，每次 +2，使列举的值始终是奇数。

let n = 100;  
let isPrime = (n === 2) || (n % 2 !== 0);  
for (let i = 3, last = Math.sqrt(n); i <= last; i += 2) {  
 if (n % i === 0) {  
 isPrime = false;  
 break;  
 }  
}  
alert(isPrime ? "质数" : "合数"); // "合数"

我们的程序可以用于处理用户输入并得到结果了，不过务必记得进行输入检查。

let n = parseInt(prompt("请输入一个大于 1 的正整数。"));  
while (true) { // 循环接受输入。  
 if (isNaN(n) || !isFinite(n) || n <= 1) {  
 alert("输入不符合要求，程序停止");  
 break; // 如果输入不符合要求，就停止循环接受输入。  
 }  
   
 let isPrime = (n === 2) || (n % 2 !== 0);  
 for (let i = 3, last = Math.sqrt(n); i <= last; i += 2) {  
 if (n % i === 0) {  
 isPrime = false;  
 break; // 这个 break 语句只退出当前所在的循环。  
 }  
 }  
 alert(`${n}是一个${isPrime ? "质数" : "合数"}`); // 使用模板字符串来拼凑信息  
 n = parseInt(prompt("请输入一个大于 1 的正整数。"));  
}

我们的质数判断程序遵循的基本流程是“输入-处理-输出”。为了避免每次输出后都要重新运行才能开始新的流程，我们可以用一个循环将流程包进去。由于我们已经认识了 break 语句，因此可以自由决定 while 循环何时终止。循环头的条件用 true 来表示“条件始终成立”，表示它不再管条件判断，只需不断进行循环以重复相同的流程。这样的程序称为“Read-Eval-Print Loop”（输入-处理-输出循环），缩写为 **REPL**。

练习 4.4.2

1. 本节提供了一个完整的用于判断质数的 REPL 示例程序，请在此基础上对它进行修改：如果 n 是一个合数，那么同时显示发现的第一个因数。
2. 对本节的示例程序进行扩充，接受一个用户输入的整数 n，查找 2 ~ n 范围内的所有质数并显示。

* （提示：将找到的质数放在数组中）

### 数组遍历

假如我们有一列排列整齐的课桌，每张课桌上都写着使用它的学生的姓名，现在我们要依次浏览并记录每张课桌上的姓名，最直观的的办法显然是：从第一张课桌开始，记录课桌上的信息，然后走到下一个课桌，以此类推。

这个过程用精确的语言描述一下：

1. 走到第一张课桌的位置，记录信息
2. 走到下一张课桌的位置，如果这里确实还有课桌，就继续记录。
3. 如果没有课桌了，就停止这个过程。
4. 否则，回到步骤 2。

我们应该怎样用 JavaScript 来实现这个过程呢？相信答案已经呼之欲出了——循环！使用循环来解决这个问题。

let queue = ["Sonam", "Susanna Kliment", "Unnr Radmila", "Davide", "Rebekah "];  
  
for (let i = 0; i < queue.length; i += 1) {  
 alert(`第${i}个学生的姓名是：${queue[i]}`);  
}  
// Sonam  
// Susanna Kliment  
// Unnr Radmila  
// Davide  
// Rebekah

我们使用 for 语句依次访问了数组中的每一个元素，变量 i 表示数组元素的*索引*，它是一个约定俗成的名称。如果这个索引值小于数组长度，说明还没有到数组尽头，那么就继续进行处理，否则就停止循环。依次访问数组每一个元素的过程称为*遍历*。