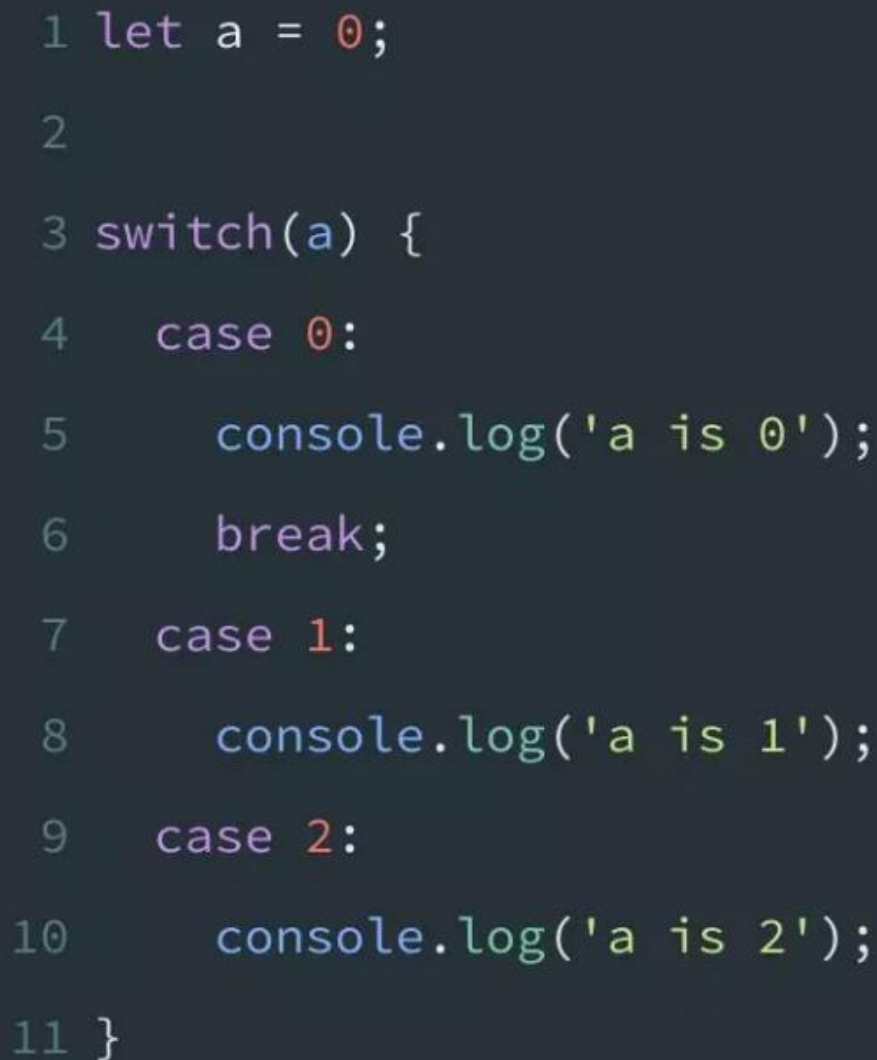


为什么程序员都不喜欢使用 switch，而是大量的 if……else if ？

请用 5 秒钟的时间查看下面的代码是否存在 bug。

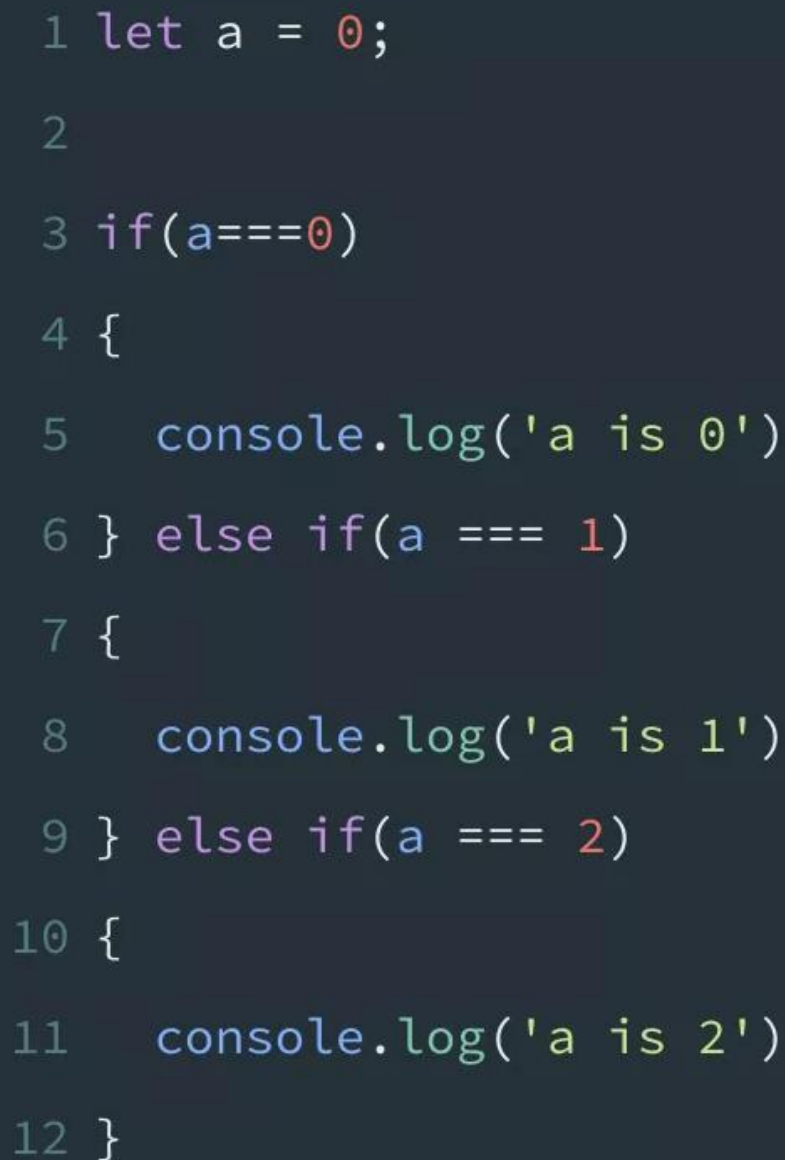


```
1 let a = 0;
2
3 switch(a) {
4   case 0:
5     console.log('a is 0');
6     break;
7   case 1:
8     console.log('a is 1');
9   case 2:
10    console.log('a is 2');
11 }
```

OK，熟练的程序猿应该已经发现 Bug 所在了，在第 8 行和第 10 行下面我没有添加关键字 **break**；这就导致这段代码的行为逻辑与我的设计初衷不符了。

缺点一. 语法正确，逻辑错误

这就是第一个理由为什么程序猿很少使用 **switch** 来做条件判断，对于新手来说忘记写 **break** 实在是再普通不过了，就算是老猿忘记写也是时有发生的事情，而这个语法错误在诸多的语法检查器上没有办法检查出来的，因为从语法角度来说是正确的！可是代码的处理逻辑却是错误的！用 **if** 来重写这段代码的话，就不会发生这种错误。



```
1 let a = 0;
2
3 if(a===0)
4 {
5   console.log('a is 0')
6 } else if(a === 1)
7 {
8   console.log('a is 1')
9 } else if(a === 2)
10 {
11   console.log('a is 2')
12 }
```

上面的代码为了保证正确我添加了 **else** 做一个逻辑上的保证，其实如果不写 **else**，这段代码也不会发生逻辑错误，而且一旦我忘记写花括号的时候，语法编译器是会提示我添加的，甚至可以使用 **eslint** 这种的工具强制我使用花括号，这样就不会犯语法错误了，一旦出现 **bug**，那么肯定是我逻辑上的问题了。

缺点二 .死板的语法

switch 尽管对于 **break** 很宽容，但是对判断条件很严苛，**case** 后面只能跟常量，如果你用 **C** 编写的话，甚至只能用 **int** 类型作为判断条件。对于我们这么潇洒自如的程序猿来说，这种限制实在是太麻烦了，用 **if** 的话，别说是常量了，我用函数都可以，真正做到方便快捷。

缺点三 .需要子函数来处理分支

这个缺点跟缺点一有关，为了防止漏写 **break**，因此建议把分支处理方法独立成一个子函数来处理，这样在阅读代码的时候就会减少忘记写 **break** 带来的 **bug**，那么用 **if** 来写的话，我想怎么写就怎么写，非常随意自由，但是这也导致了代码的可读性大大降低。

switch 的优点

既然 **switch** 有这么严重的缺点，那怎么在所有语言中依然会存在呢？那就说下 **switch** 的优点吧，它的优点也刚好是它的缺点。

在很久很久以前，那时候的电脑性能还不如一台小霸王学习机的时候，聪明的计算机科学家为了提高计算机的处理速度，将一些逻辑分支处理方法简化了一下，把一些需要做逻辑判断的操作给固定死，然后只要查表一样一个一个对一下就能做出相应的反应了。

比如说 **a=0** 的判断，**switch** 和 **if** 在 **cpu** 上面的处理方式是不同的，**switch** 是在编译阶段将子函数的地址和判断条件绑定了，只要直接将 **a** 的直接映射到子函数地址去执行就可以了，但是 **if** 处理起来就不一样了。

它首先要将 **a** 的值放到 **CPU** 的寄存器中，然后要把比较的值放到 **CPU** 的另一个寄存器中，然后做减法，然后根据计算结果跳转到子函数去执行，这样一来就要多出 3 步的操作了，如果逻辑判断多的话，那么将会比 **switch** 多许多倍的操作，尽管寄存器操作的速度很快，但是对于当时的学习机来说，这点速度根本不够用啊。

那还有一个问题，为什么要使用 **break** 来做一个判断结束呢？这不是很容易造成语法错误了？那就要说到子函数的问题上了。

在早起的电脑代码中是没有子函数的概念的，那时候都是用 **goto** 随意跳转的，你想去第 10 行代码，很简单 **goto 10** 就可以了。这种编程思维在 **C** 的早期阶段还是一直受到影响的，因此早期的 **C** 也没有子函数，都是一堆逻辑处理混乱在一起，**goto** 满天飞，所以那时候你没有一个最强大脑是写不了程序的。那为了告诉程序我这里条件判断处理结束，就添加了 **break** 作为终止符号。后来慢慢的有了子程序，有了更好的编程规范，才一步一步的将写代码沦落到体力劳动。

后来发展的新语言为了标榜自己的血统，多少都要参考下 C，然后就把 **switch** 这种诡异的语法也继承下来了。但是也不是所有的语言都照搬，比如 Google 发明的新语言 **golang** 和 **kotlin** 就又把 **switch** 包装了一下，去掉了令人误会的语法，又让 **switch** 变得灵活起来了，对了，在代码重构的时候，还是用 **switch** 把，这样看起来的确代码更简洁哦！

