### JavaScript 注释语句操作

被注释的代码块在JavaScript之中是不会被运行的。注释是一个非常好的方式可以让你自己以及其他人明白这段代码是怎么运行的。

JavaScript中的注释方式有以下两种：

使用 // 通知JavaScript来忽略当前行的代码

// This is an in-line comment.

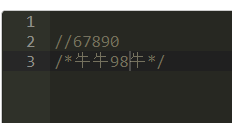
你也可以使用多行注释来注释你的代码，以/\*开始，通过\*/来结束，参考如下代码：

/\* This is a   
multi-line comment \*/

**最佳实践**  
在编写代码时，应该通过添加注释来让你的代码看起来更加地清晰易懂。良好的注释能够清晰地传达你写的代码的意图，便于你或者他人在阅读你的代码时能更快的理解这段代码的意图。

#### 任务

尝试创建这两种类型的注释。



### 声明JavaScript变量

在计算机科学中, data(数据)大于一切，因为它对于计算机的意义重大。JavaScript提供七种不同的data types(数据类型)，它们是undefined（未定义）, null（空）, boolean（布尔型）, string（字符串）, symbol(符号), number（数字）, and object（对象）。

举个栗子, 计算机能够分辨不同的数字, 例如数字 12，但是"12"和"dog", 或"123 cats", 却是字母的集合。 计算机能够精确地运算数字， 但是无法处理字符串中的数字。

Variables（变量）允许计算机以动态的形式来存储和操作数据，通过操作指向数据的指针而不是数据本身来避免了内存泄露，以上的七种数据类型都可以存储到一个Variables（变量）中。

Variables（变量 ）类似于您在数学中使用的x和y变量，这意味着它们通过一个简单的名称来表示我们要引用的数据。 计算机中的Variables（变 量）与数学变量不同，因为它们可以在不同时间存储不同的值。

通过在变量的前面使用关键字var，我们告诉 JavaScript 来创建或者 declare（声明）一个变量,就像这样：

var ourName;

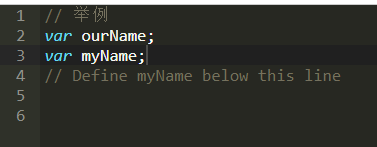
上面代码的意思是创建一个名为ourName的variable（变量），在JavaScript中我们使用;（分号）来结束一段声明。

Variable （变量）的名字可以由数字、字母、$ 或者 \_组成，但是不能包含空格或者以数字为首。

#### 任务

使用var 关键字来创建一个名为myName的变量。

**提示**   
如果遇到困难了，请看下ourName的例子是怎么写的。



### JavaScript 使用=为变量赋值

在JavaScript中，您可以使用=赋值运算符将值存储在变量中。

myVariable = 5;

把Number数字5赋给变量myVariable。

赋值过程是从右向左进行的。所有 = 赋值运算符右边的值都会被赋到左边的变量中。

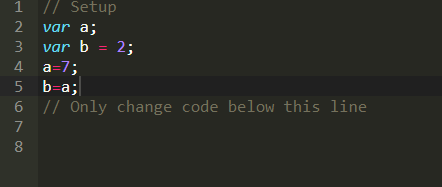
myVar = 5;   
myNum = myVar;

数值 5 被赋给变量 myVar 中， 然后变量 myVar 又赋给变量 myNum ，这样子 myNum 变量中的值也是 5了。

#### 任务

把数值 7 赋给变量 a。

把变量 a 中的内容赋给变量 b。



### JavaScript 使用=为变量赋初始值

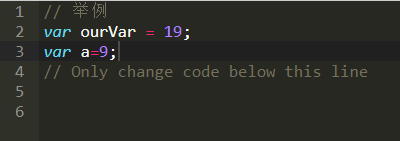
通常地我们会在initialize初始化声明变量的时候就会给变量赋一个初始值。

var myVar = 0;

创建一个名为 myVar 的变量并指定初始值为 0。

#### 任务

通过关键字 var 定义一个变量 a 并且指定初始值为 9。

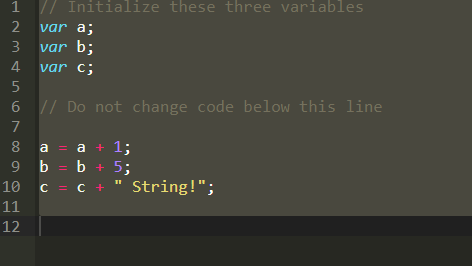


### 理解JavaScript未定义变量

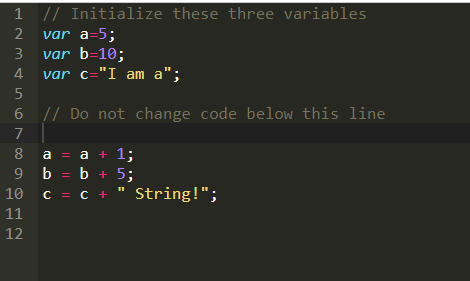
当 JavaScript 中的变量被声明的时候，它们的初始值为 undefined。如果对一个值为 undefined 的变量进行运算操作的时候，那么结果将会是 NaN，NaN 的意思是 "Not a Number"。如果将一个没有 定义 的变量跟字符串进行连接的时候，你将获得一个**字符串**"undefined"。

#### 任务

定义3个变量 a、b、c ，并且分别给它们赋值：5、10、"I am a" ，这样它们就不会变成 undefined（未定义的了）。

原：

结果：



### 理解JavaScript变量的大小写敏感性

在 JavaScript 中所有的变量都是大小写敏感的。这意味着你要区分大写字母和小写字母。

MYVAR与MyVar和myvar 是截然不同的变量。这就有可能导致多个不同的变量却有着有相似的名字。正是由于以上原因所以强烈地建议你, 不要使用这一特性。（尽量避免设置名字相近难以区分的变量名）

#### 最佳实践

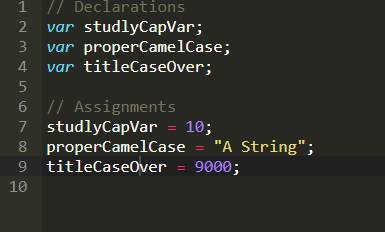
使用 驼峰命名法 来设置一个 Javascript 变量，在 驼峰命名法 中，变量名的第一个单词的首写字母小写，后面的单词的第一个字母均为大写。

**举个栗子:**

var someVariable;  
var anotherVariableName;  
var thisVariableNameIsTooLong;

#### 任务

修正声明的变量，使它们的命名符合 驼峰命名法 的规范。



### JavaScript加法运算

Number 是JavaScript中的数据类型，表示数字数据。

现在让我们尝试在JavaScript中做加法运算。

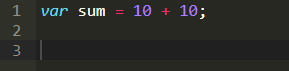
JavaScript 中使用 + 运算符来让两个数字执行加法运算。

#### 举个栗子

myVar = 5 + 10; // 等于 15

#### 任务

改变数字 0 让变量 sum 的值为 20.



### JavaScript减法运算

我们可以在 JavaScript 中执行减法运算。

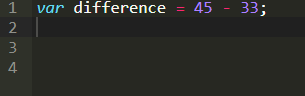
JavaScript 中使用 - 运算符来做减法运算。

**举例**

myVar = 12 - 6; // 等于 6

#### 任务

改变数字 0 让变量 difference 的值为 12。



### JavaScript乘法运算

我们可在 JavaScript 中使用乘法运算。

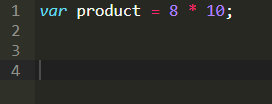
JavaScript 中可以使用 \* 运算符来让两个数字相乘。

#### 举个栗子

myVar = 13 \* 13; // assigned 169

#### 任务

改变数值 0 来让变量 product 的值等于 80。



### JavaScript除法运算

我们可以在 JavaScript 中做除法运算。

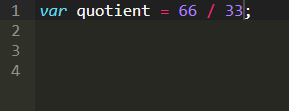
JavaScript 中使用 / 运算符做除法运算。

#### 举个栗子

myVar = 16 / 2; // assigned 8

#### 任务

改变数值0来让变量 quotient 的值等于2。



### JavaScript自增运算++

使用 ++ 运算符，我们可以轻易地对变量进行自增或者+1运算。

i++;

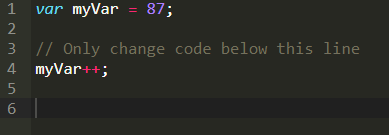
等效于

i = i + 1;

**提示**   
i++;这种写法，省去了使用=运算符的必要。

#### 任务

重写代码，使用 ++ 来对变量 myVar 进行自增操作。



### JavaScript自减运算--

使用自减运算符符号 -- ，你可以轻易地对一个变量执行 自减 或者-1操作。

i--;

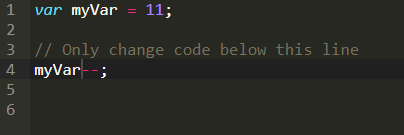
等效于

i = i - 1;

**提示**   
i--;这种写法，省去了使用=运算符的必要。

#### 任务

重写代码，使用 -- 对 myVar 执行自减操作。



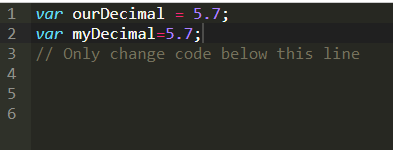
### JavaScript浮点数

我们可以把小数存储到变量中。小数也被称作 浮点数 。

**提示**  
不是所有的实数都可以用 浮点数 来表示。因为可能存在四舍五入的错误，[详情在这儿](https://en.wikipedia.org/wiki/Floating_point#Accuracy_problems)。

#### 任务

创建一个变量 myDecimal 并给它赋值一个浮点数。 (e.g. 5.7)。



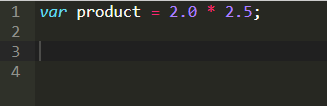
### JavaScript小数乘法运算

在 JavaScript 中，你可以用小数进行计算，就像整数一样。

让我们把两个小数相乘，并得到它们相乘的结果。

#### 任务

改变 0.0 的数值让变量product的值等于 5.0。

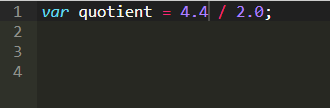


### JavaScript小数的除法运算

现在让我们来用一个小数做除法运算。

#### 任务

改变数值 0.0 的值让变量 quotient 的值等于 2.2.



### JavaScript取余运算%

使用%运算符来对两个数字进行**求余**运算。

#### 举个栗子

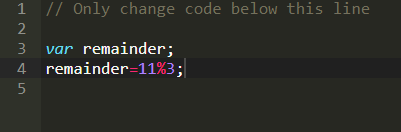
5 % 2 = 1

**用法**   
在数学中，看一个数是奇数还是偶数，只需要看这个数除以2得到的余数是0还是1。

17 % 2 = 1 (17 is 奇数)   
48 % 2 = 0 (48 is 偶数)

#### 任务

使用 % 操作符，计算11除以3剩下的余数，并把余数赋给变量remainder。



### JavaScript +=赋值操作

在编程当中，通常通过赋值来修改变量的内容。请记住，先计算=右边，然后把计算出来的结果赋给左边。

myVar = myVar + 5;

以上是最常见的运算赋值语句，先运算、再赋值。

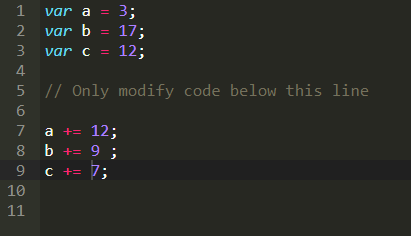
还有一类操作符是一步到位合并了运算与赋值的操作。

这类运算符的其中一种就是 += 运算符。

myVar += 5; 也是把数值 5 加到变量 myVar上。

#### 任务

使用 += 操作符实现同样的效果。



### avaScript -=赋值操作

与 += 操作符类似，-= 操作符用来对一个变量进行减法赋值操作。

myVar = myVar - 5;

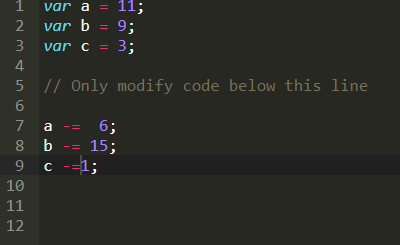
将会从变量 myVar 中减去数值 5。

也可以写成这种形式：

myVar -= 5;

#### 任务

使用 -= 操作符实现同样的效果。



### JavaScript \*=赋值操作

\*= 运算符是让变量与一个数相乘并赋值。

myVar = myVar \* 5;

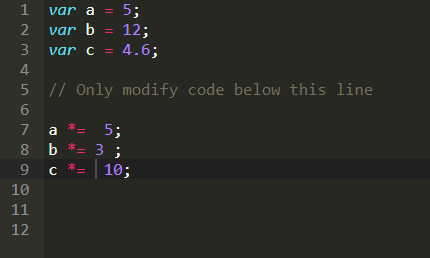
将会把变量 myVar 与数值 5相乘并赋值。

也可以写作这样的形式:

myVar \*= 5;

#### 任务

使用 \*=运算符实现同样的效果。



### JavaScript /=赋值操作

/= 运算符是让变量与另一个数相除并赋值。

myVar = myVar / 5;

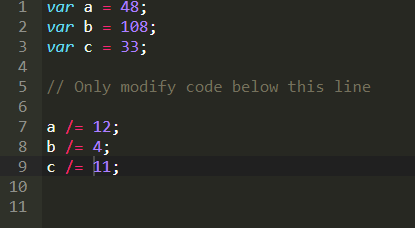
会把变量 myVar 的值除于 5并赋值。

也可以写作这样的形式:

myVar /= 5;

#### 任务

使用 /=操作符实现同样的效果。

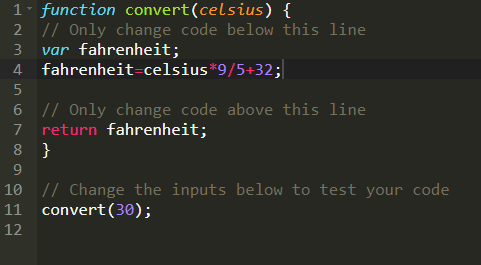


### JavaScript 基本预算综合练习

为了测试你的学习效果，我们结合之前的知识点来做一个摄氏度转华氏度的小程序。

从Celsius摄氏度转换为Fahrenheit华氏度的算法是：摄氏度的温度乘于9除于5，再加上32。

创建一个变量 fahrenheit，然后计算出摄氏度对应的华氏度。



### JavaScript 声明字符串变量

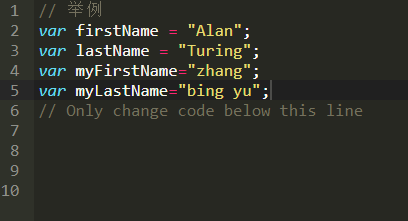
先前我们使用过的代码：

var myName = "your name";

"your name" 被称作 字符串。 字符串是用单引号或双引号包裹起来的一连串的零个或多个字符。

#### 任务

创建两个新的 字符串 变量：myFirstName 和 myLastName 分别为它们赋上你的姓和名的值。



### JavaScript 转义字符串中的引号

当你定义一个字符串时，你必须以单引号或双引号作为开头和结尾。那么当你需要在字符串中使用一个: " 或者 '时该怎么办呢?

在 JavaScript 中，你可以通过在引号前面使用 反斜杠(\) 来转义引号。

var sampleStr = "Alan said, \"Peter is learning JavaScript\".";

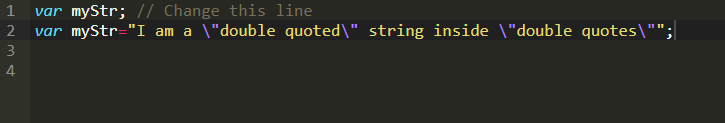
这标志着提示 JavaScript 该单引号或双引号并不是字符串的结尾，而是出现在字符串内的字符。所以，如果你要打印字符串到控制台，你将得到：

Alan said, "Peter is learning JavaScript".

#### 任务

使用 反斜杠 将一个字符串以双引号包裹并赋值给变量 myStr，当你将它打印到控制台的时候，你会看到：

I am a "double quoted" string inside "double quotes"



### JavaScript 使用单引号引用字符串

在 JavaScript 中的 字符串 要用单引号或双引号来包裹它，只要你在开始和结束都使用相同类型的引号，单引号和双引号的功能在JavaScript中是相同的。

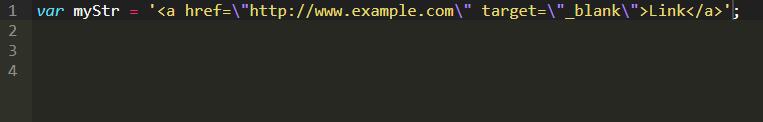
"This string has \"double quotes\" in it"

当我们需要在字符串中使用与开头结尾相同的引号时，我们需要对引号进行 转义 。如果你有很多双引号的字符串，使用转义字符可能导致难以阅读。这时候可以使用单引号。

'This string has "double quotes" in it. And "probably" lots of them.'

#### 任务

更改myStr字符串用于开始和结束的双引号为单引号，并移除转义符号。



### JavaScript 转义字符串中的特殊符号

字符串中的转义序列

引号不是字符串中唯一的可以被转义字符。下面是常见的转义序列列表:

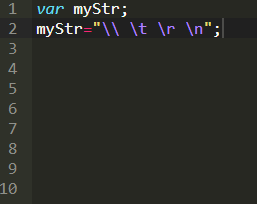
| **Code** | **Output** |
| --- | --- |
| \' | 单引号 |
| \" | 双引号 |
| \\ | 反斜杠符 |
| \n | 换行符 |
| \r | 回车符 |
| \t | 制表符 |
| \b | 退格符 |
| \f | 换页符 |

注意，如果你想要显示一个反斜杠就必须要转义它。

#### 任务

按照下面的顺序，并且用空格来分割：   
反斜杠 制表符 回车符 换行符 并把它赋值给变量 myStr

注意：请使用双引号包含内容，并且注意内容前后不要留有非必要的空格



### JavaScript字符串连接方式

在 JavaScript 中，当 + 运算符与 字符串 一起使用的时候，它被称作 连接 运算符。你可以通过和其他字符串连接 来创建一个新的字符串。

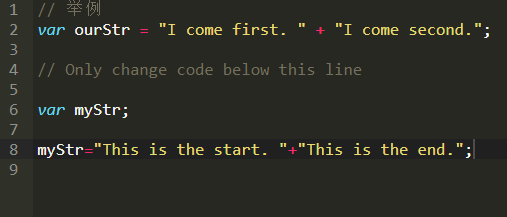
**举个栗子**

'My name is Alan,' + ' I concatenate.'

**注意**  
注意空格。连接操作不会添加两个字符串之外的空格，所以想加上空格的话，你需要自己在字符串里面添加。

#### 任务

使用 + 运算符，把字符串 "This is the start. " 和 "This is the end." 连接起来并赋值给变量 myStr。



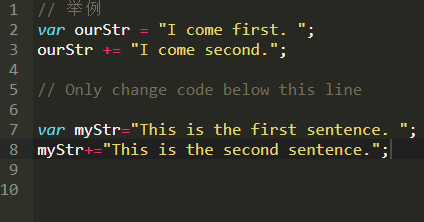
### JavaScript +=连接字符串操作

我们还可以使用 += 运算符来将字符串 连接 到现有字符串变量的末尾。对于那些非常长的字符串来说，这是十分有用的。

**注意**  
注意空格。连接操作不会添加两个字符串以外的空格，所以如果想要加上空格的话，你需要自己在字符串里面添加。

#### 任务

通过使用 += 操作符来连接这两个字符串:  
"This is the first sentence. " 和 "This is the second sentence." 并赋给变量 myStr。



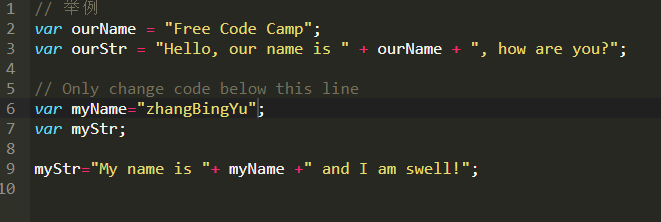
### JavaScript 使用变量连接字符串

有时候你需要创建一个填字风格的字符串。

通过使用连接运算符 + ，你可以插入一个或多个变量来组成一个字符串。

#### 任务

把你的名字赋值给变量 myName ，然后把变量 myName 插入到字符串 "My name is " 和 " and I am swell!"之间，并把连接后的结果赋值给变量 myStr。

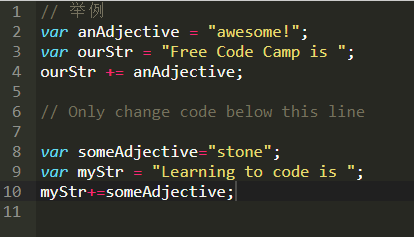


### JavaScript 追加变量到字符串上

我们不仅可以创建出多行的字符串，还可以使用加等号(+=)运算符来追加变量到字符串上。

#### 任务

设置变量 someAdjective 的值，并使用 += 运算符把它追加到变量 myStr 上。



### JavaScript 获取字符串的长度

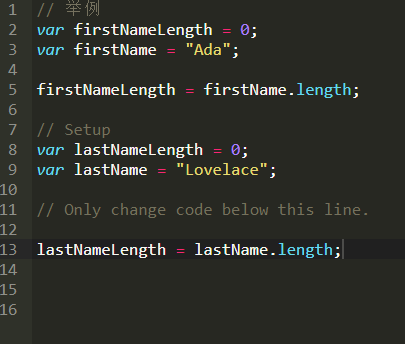
你可以通过在字符串变量或字符串后面写上 .length 来获得变量中string （字符串）值的长度。

"Alan Peter".length; // 10

例如，我们创建了一个变量 var firstName = "Charles"，我们就可以通过使用 firstName.length 来获得 "Charles" 字符串的长度。

#### 任务

使用 .length 属性来获得变量 lastName 的长度，并把它赋值给变量 lastNameLength。



### JavaScript 使用中括号获取字符串中的第一个字符

括号符号是一种在字符串中的特定索引处获取字符的方法。

() ：小括号。

[] ：中括号

{}：大括号

JavaScript中只有字符串类型，没有字符类型。

如果要获取字符串中的某个字符，我们需要通过[索引]来获得对应的字符。

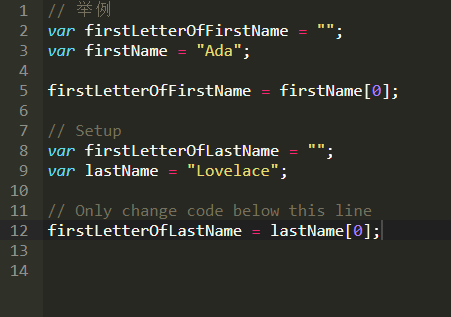
大多数现代编程语言，如JavaScript，不同于人类从1开始计数。它们是从0开始计数，这被称为 基于零 的索引。

例如, 在单词 "Charles" 中索引0上的字符为 "C"，所以在 var firstName = "Charles" 中，你可以使用 firstName[0] 来获得第一个位置上的字符。

#### 任务

使用[索引]来得到变量 lastName 中的第一个字符，并赋给变量 firstLetterOfLastName 。

**提示**   
如果你遇到困难了，可以参考一下变量 firstLetterOfFirstName 是如何赋值的。



### 理解JavaScript字符串的不可变性

在 JavaScript 中，字符串 的值是 不可变的，这意味着一旦字符串被创建就不能被改变。

例如，下面的代码：

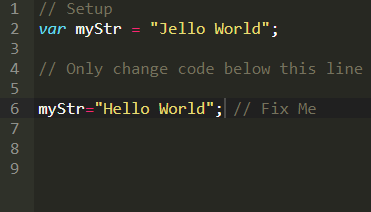
var myStr = "Bob";  
myStr[0] = "J";

是不会把变量 myStr 的值改变成 "Job" 的，因为变量 myStr 是不可变的。注意，这 **并不** 意味着 myStr 永远不能被改变，只是字符串字面量 string literal 的各个字符不能被改变。改变 myStr 中的唯一方法是重新给它赋一个值，就像这样：

var myStr = "Bob";  
myStr = "Job";

#### 任务

把myStr 的值改为Hello World。



### JavaScript 使用中括号索引查找字符串中的第N个字符

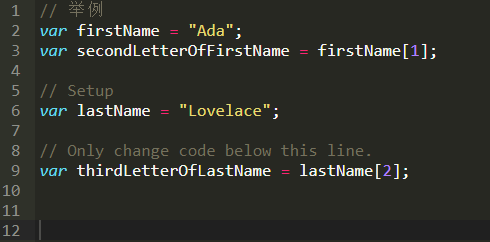
你也可以使用 [索引]来获得一个字符串中的某个位置的字符。

请记住，计算机是从 0 开始计数，所以获取第一个字符的索引实际上是[0]。

#### 任务

让我们使用[索引]，把 lastName 变量的第三个字符赋值给 thirdLetterOfLastName。

**提示**  
如果你遇到困难了，可以参考一下 secondLetterOfFirstName 变量是如何做的。



### JavaScript 使用中括号索引查找字符串中的最后一个字符

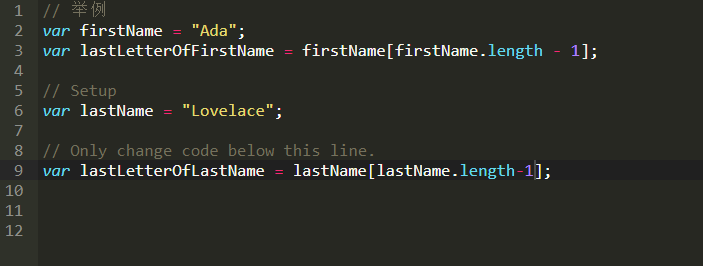
为了得到一个字符串的最后一个字符，你可以获取该字符串的长度并减一来作为最后一个字符的索引。

例如，在 var firstName = "Charles" 中，你可以这样操作 firstName[firstName.length - 1] 来得到字符串的最后的一个字符。

#### 任务

使用 [索引] 来取得 lastName 变量中的最后一个字符。

**提示**  
如果你遇到困难了，可以参考一下lastLetterOfFirstName变量上是怎么做的。



### JavaScript 使用中括号索引查找字符串中倒数第N个字符

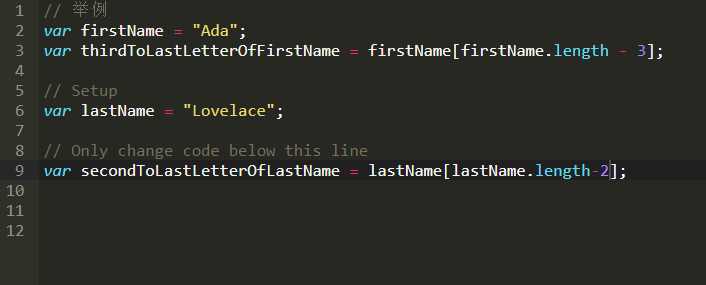
我们既然可以获取字符串的最后一个字符，也可以用相同的方法获取字符串的倒数第N个字符。

例如，你可以这样 firstName[firstName.length - 3] 操作来获得 var firstName = "Charles" 字符串中的倒数第三个字符。

#### 任务

使用 [索引] 来获得 lastName 字符串中的倒数第二个字符。

**提示**  
如果你遇到困难了，可以参考一下thirdToLastLetterOfFirstName 变量是如何做到的。



### Javascript 字符串操作综合训练

**填词造句**

现在我们将结合之前学习的字符串相关知识，来玩一个填字风格的文字游戏。

通过使用提供的变量：名词myNoun、形容词myAdjective、动词myVerb、副词myAdverb，来创建一个新的字符串并赋值到 result。

此外，除了所提供的词之外，你还需要添加额外的连接字符串。

例如：result = myNum+' aa '+myNun;使用+连接字符串，并且中间用额外字符" aa "来隔开，注意间隔字母两边要用空格，你也可以使用，来分割

我们提供了多种审核，该审核用多个不同单词的输入来运行你的函数，以确保所有提供的单词和你额外添加的连接字符串能够准确输出。

wordBlanks("","","","") 应该返回一个字符串.

wordBlanks("dog", "big", "ran", "quickly") 应该包含所有传入的单词并且单词之间要有间隔(可以添加任意修饰的单词).

wordBlanks("cat", "little", "hit", "slowly") 应该包含所有传入的单词并且单词之间要有间隔(可以添加任意修饰的单词).



### Javascript 数组操作

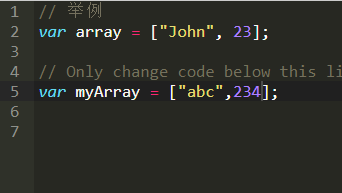
通过 数组，我们可以将多个数据存储于同一个变量中。

通过左括号[开始定义一个数组，以右括号]结束定义，并把每个数据条目之间用逗号隔开，就像这样：  
var sandwich = ["peanut butter", "jelly", "bread"]。

#### 任务

创建一个包含 字符串 和 数字 的数组 myArray。

**提示**  
如果你遇到困难，请参考文本编辑器中的示例代码。

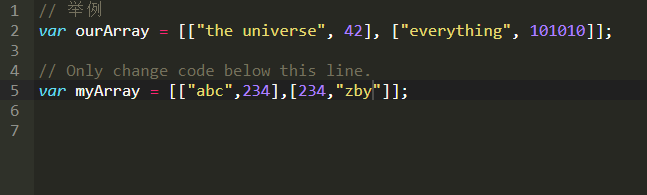


### Javascript 多维数组操作

你也可以在数组中包含其他数组，就像这样: [["Bulls", 23], ["White Sox", 45]]。这被称为一个 多维数组。

#### 任务

创建一个名为 myArray的多维数组。



# JavaScript 使用索引查找数组中的数据

我们可以像操作字符串一样通过数组索引[index]来访问数组中的数据。

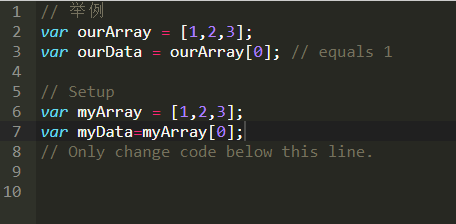
数组索引的使用与字符串索引一样，不同的是，通过字符串的索引得到的是一个字符，通过数组索引得到的是一个数据条目。与字符串类似，数组也是 基于零 的索引，因此数组的第一个元素的索引是 0。

**例如**

var array = [1,2,3];  
array[0]; // 等于 1  
var data = array[1];// 等于 2

#### 任务

创建一个名为 myData 的变量，并把 myArray 的第一个索引上的值赋给它。



### JavaScript 使用索引修改数组中的数据

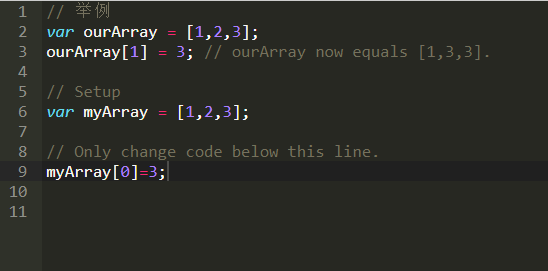
与字符串的数据不可变不同，数组的数据是可变的，并且可以自由地改变。

**例如**

var ourArray = [3,2,1];  
ourArray[0] = 1; // ourArray等于 [1,2,1]

#### 任务

修改数组 myArray 中索引0上的值为 3。



### JavaScript 使用索引操作多维数组

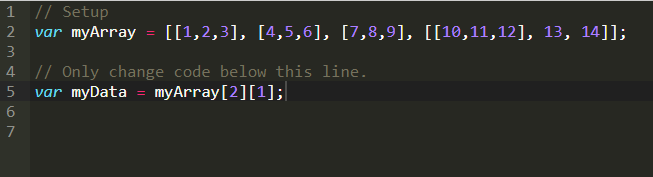
可以把 多维 数组理解成是一个 **数组中的数组**。当使用[]去访问数组的时候，第一个[index]访问的是该数组中的第N个子数组，第二个[index]访问的是第N个子数组的第N个元素。

**例如**

var arr = [  
[1,2,3],  
[4,5,6],  
[7,8,9],  
[[10,11,12], 13, 14]  
];  
arr[0]; // 等于 [1,2,3]  
arr[1][2]; // 等于 6  
arr[3][0][1]; // 等于 11

#### 任务

对数组 myArray 使用[index]操作使得 myData 的值为 8



### JavaScript push()函数追加数组数据

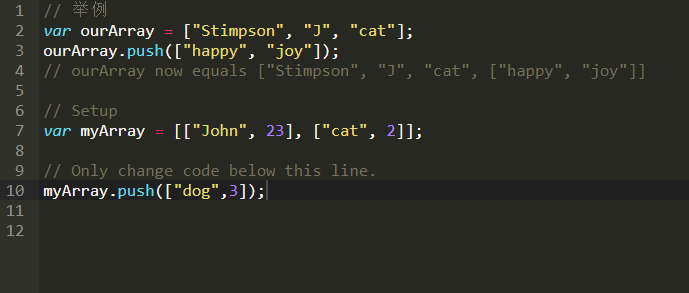
将数据追加到一个数组末尾的最简单的方法是通过 push()函数。

.push() 允许有一个或多个参数，并把它“push”到数组的末尾。

var arr = [1,2,3];  
arr.push(4);  
// 现在arr的值为 [1,2,3,4]

#### 任务

把 ["dog", 3] “push”到 myArray 变量的末尾。



### JavaScript pop()函数弹出数组最后数据

改变数组中数据的另一种方法是用 .pop() 函数。

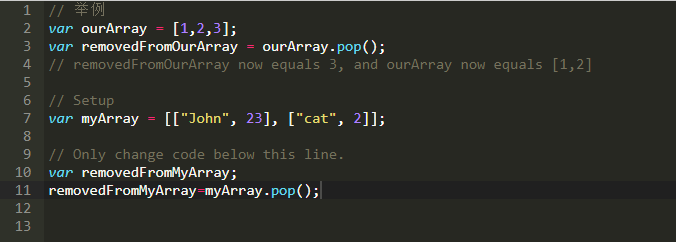
.pop() 函数用来“抛出”一个数组末尾的值。我们可以把这个“抛出”的值赋给一个变量存储起来。

数组中任何类型的数据条目（数值，字符串，甚至是数组）可以被“抛出来” 。

举个例子, 对于这段代码  
var oneDown = [1, 4, 6].pop();  
现在 oneDown 的值为 6 ，数组变成了 [1, 4]。

#### 任务

使用 .pop() 函数移除 myArray 中的最后一条，并且把“抛出”的值赋给 removedFromMyArray。



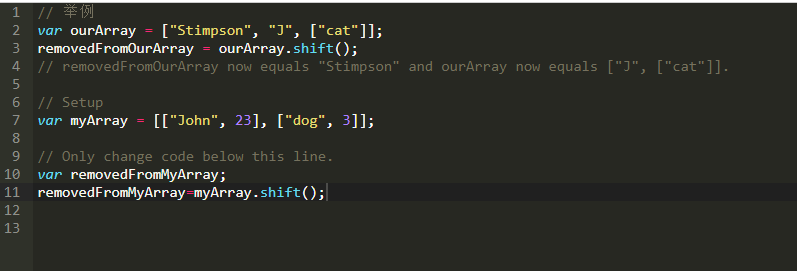
### JavaScript shift()函数移出数组第一个数据

pop() 函数用来移出数组中最后一个元素。如果想要移出第一个元素要怎么办呢？

.shift() 就是专门用来处理这类型需求的。它的工作原理类似 .pop()，但它移除的是第一个元素，而不是最后一个。

#### 任务

使用 .shift() 函数移出 myArray 中的第一项，并把“移出”的值赋给 removedFromMyArray。



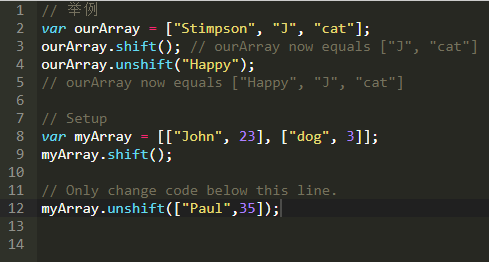
### JavaScript unshift()函数移入数据到数组第一位

你不仅可以 shift（移出）数组中的第一个元素，你也可以 unshift（移入）一个元素到数组的头部。

.unshift() 函数用起来就像 .push() 函数一样, 但不是在数组的末尾添加元素，而是在数组的头部添加元素。

#### 任务

使用 unshift() 函数把["Paul",35] 加入到 myArray 的头部。



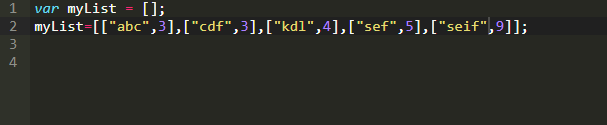
### JavaScript 创建购物清单实战案例

创建一个名叫 myList的购物清单，清单的数据格式必须是多维数组。

每个子数组中的第一个元素应该是购买的物品名称，第二个元素应该是物品的数量，类似于：

["Chocolate Bar", 15]

任务：你的购物清单至少应该有5个子数组。



### JavaScript 函数定义

在 JavaScript 中，我们可以把代码的重复部分抽取出来，放到一个函数（functions）中。

这是一个函数（function）的例子：

function functionName() {  
console.log("Hello World");  
}

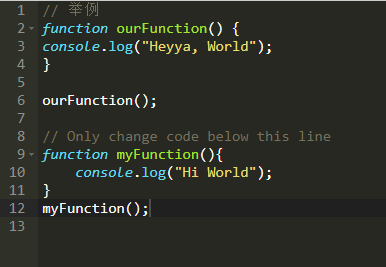
你可以通过函数名称functionName加上后面的小括号来调用这个函数（function），就像这样：

functionName();

每次调用函数时它会打印出消息的“Hello World”到开发的控制台上。所有的大括号之间的代码将在每次函数调用时执行。

#### 任务

1. 创建一个名为 myFunction 的函数，这个函数可以打印“Hi World”到开发控制台上。
2. 调用这个函数。



### JavaScript 带参数函数定义

函数的参数parameters在函数中充当占位符(也叫形参)的作用，参数可以为一个或多个。调用一个函数时所传入的参数为实参，实参决定着形参真正的值。

这是带有两个参数的函数， param1 和 param2：

function testFun(param1, param2) {  
console.log(param1, param2);  
}

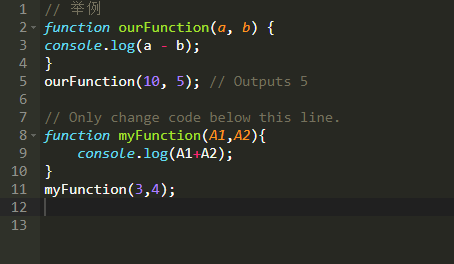
接着我们调用 testFun：

testFun("Hello", "World");

我们传递了两个参数， "Hello" 和 "World"。在函数内部，param1 等于“Hello”，param2 等于“World”。请注意，testFun 函数可以多次调用，每次调用时传递的参数会决定形参的实际值。

#### 任务

1. 创建一个名为 myFunction 的函数，它可以接收两个参数，计算参数的和，将结果输出到控制台。
2. 调用这个函数。



### JavaScript 函数全局变量定义

在 JavaScript 中， 作用域 影响着变量的作用范围。在函数外定义的变量具有 全局 作用域。这意味着，具有全局作用域的变量可以在代码的任何地方被调用。

没有使用var关键字定义的变量，会被自动创建在全局作用域中，变成全局变量。当在代码其他地方无意间定义了一个变量，刚好变量名与全局变量相同，这时候会产生意想不到的后果，可能导致你的代码出现BUG。因此你应该总是使用var关键字来声明你的变量。

#### 任务

在函数外声明一个 全局 变量 myGlobal，并给它一个初始值 10

在函数 fun1 的内部，**不**使用 var 关键字来声明 oopsGlobal，并赋值为 5。



### JavaScript 函数局部变量定义

在一个函数内声明的变量，以及该函数的参数都是局部变量，意味着它们只在该函数内可见。

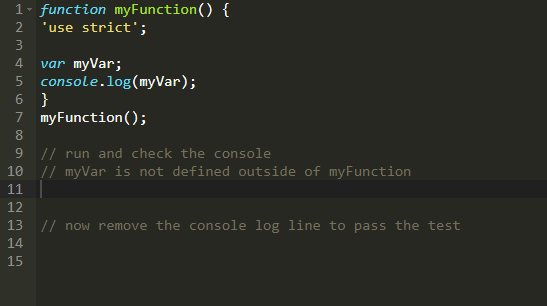
这是在函数 myTest内声明局部变量loc 的最佳例子：

function myTest() {  
var loc = "foo";  
console.log(loc);  
}  
myTest(); // "foo"  
console.log(loc); // "undefined"

在函数外，loc 是未定义的。

#### 任务

在函数 myFunction 内部声明一个局部变量 myVar，并删除外部console.log。



### JavaScript 函数全局变量与局部变量差异

一个程序中有可能具有相同名称的 局部 变量 和 全局 变量。在这种情况下，局部 变量将会优先于 全局 变量。

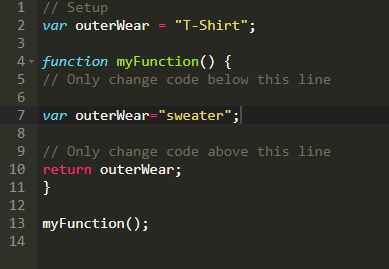
下面为例：

var someVar = "Hat";  
function myFun() {  
var someVar = "Head";  
return someVar;  
}

函数 myFun 将会返回 "Head"，因为 局部变量 优先级更高。

#### 任务

给 myFunction 添加一个局部变量来覆盖 outerWear 的值为 "sweater"。



### JavaScript 函数使用return返回值

我们可以把数据通过函数的 参数 来传入函数，也可以使用 return 语句把数据从一个函数中传出来。

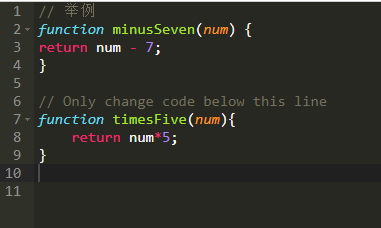
**举个栗子**

function plusThree(num) {  
return num + 3;  
}  
var answer = plusThree(5); // 8

plusThree 带有一个为 num 的 参数 并且返回（returns）一个等于 num + 3 的值。

#### 任务

创建一个函数 timesFive 接受一个参数, 把它乘于 5 之后并返回（returns）。



### JavaScript 函数使用返回值进行赋值

复习一下，赋值之前，先完成等号右边的操作。这意味着我们可把一个函数的返回值，赋值给一个变量。

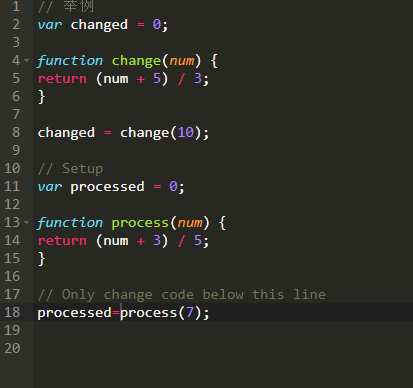
假设我们预先定义的函数 sum 其功能就是将两个数字相加，那么：

ourSum = sum(5, 12);

将调用 sum 函数，返回return了一个数值 17，然后把它赋值给了 ourSum 变量。

#### 任务

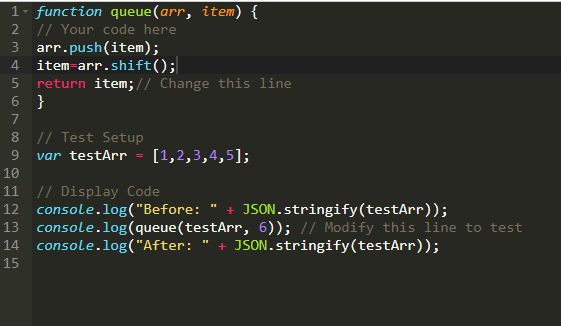
调用 process 函数并传入参数值 7，然后把返回的值赋值给变量 processed。



### JavaScript 队列

在计算机科学中 队列（queue）是一个抽象的数据结构，队列中的数据条目都是有秩序的。新的条目会被加到 队列 的末尾，旧的条目会从 队列 的头部被移出。

写一个函数 queue ，用一个数组arr和一个数字item作为参数。数字item添加到数组的结尾，然后移出数组的第一个元素，最后队列函数应该返回被删除的元素。



### JavaScript 布尔boolean数据类型

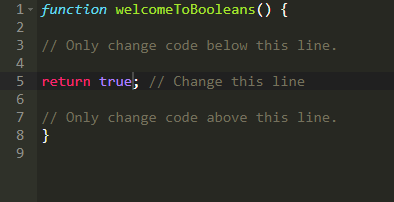
布尔（Boolean）是另一种数据类型

。布尔 值要么是true 要么是false。它非常像电路开关， true 是“开”，false 是“关”。这两种状态是互斥的。

**注意**  
Boolean 值绝不会写作被引号包裹起来的形式。字符串 的 "true" 和 "false" 不是 布尔值，在 JavaScript 中也没有特殊含义。（布尔值不是字符串）

#### 任务

修改 welcomeToBooleans 函数，让它返回 true 而不是 false。



### JavaScript if语句

If 语句用于在代码中做条件判断。关键字 if 告诉 JavaScript 在小括号中的条件为真的情况下去执行定义在大括号里面的代码。这种条件被称为 Boolean 条件，因为他们只可能是 true（真）或 false（假）。

当条件的计算结果为 true，程序执行大括号内的语句。当布尔条件的计算结果为 false，大括号内的代码将不会执行。

**伪代码**

if(*条件为真*){  
*语句被执行*  
}

**示例**

function test (myCondition) {  
if (myCondition) {  
return "It was true";  
}  
return "It was false";  
}  
test(true);// returns "It was true"  
test(false); // returns "It was false"

当 test 被调用，并且传递进来的参数值为 true，if 语句会计算 myCondition 的结果，看它是真还是假。如果条件为 true，函数会返回 "It was true"。当 test 被调用，并且传递进来的参数值为 false，myCondition 不 为 true，并且不执行大括号后面的语句，函数返回 "It was false"。

#### 任务

在函数内部创建一个 if 语句，如果该参数 wasThatTrue 值为 true，返回 "That was true"，否则，并返回 "That was false"。



### JavaScript 等于比较运算符==

在 JavaScript 中，有很多 相互比较的操作。所有这些操作符都返回一个 true 或 false 值。

最基本的运算符是相等运算符：==。相等运算符比较两个值，如果它们是同等，返回 true，如果它们不等，返回 false。b必须注意相等运算符不同于赋值运算符（=），赋值运算符是把等号右边的值赋给左边的变量。

function equalityTest(myVal) {  
if (myVal == 10) {  
return "Equal";  
}  
return "Not Equal";  
}

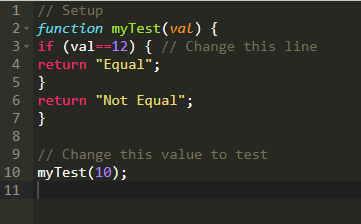
如果 myVal 等于 10，相等运算符会返回 true，因此大括号里面的代码会被执行，函数将返回 "Equal"。否则，函数返回 "Not Equal"。

在 JavaScript 中，为了让两个不同的 数据类型（例如 数字 和 字符串）的值可以作比较，它必须把一种类型转换为另一种类型。然而一旦这样做，它可以像下面这样来比较：

1 ==1// true  
1 ==2// false  
1 == '1' // true  
"3"==3// true

#### 任务

把 相等运算符 添加到指定的行，这样当 val 的值为12的时候，函数会返回"Equal"。



### JavaScript 全等于比较运算符===

全等（===）是相对于相等操作符（==）的一种操作符。与相等操作符不同的是全等比较严格，它会同时比较元素的值和 数据类型。

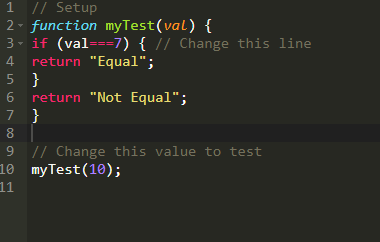
**举个例子**

3 === 3 // true  
3 === '3' // false

3 是一个 数字 类型的，而'3' 是一个 字符 类型的，所以3不全等于'3'。

#### 任务

在 if 语句值使用全等运算符，这样当 val 严格等于7的时候，函数会返回"Equal"。



### avaScript 不等于比较运算符!=

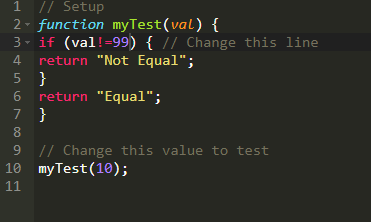
不相等运算符（!=）与相等运算符是相反的。在不相等运算符中，如果条件结果在相等运算符中为“false”则会被判断为返回 true。与相等运算符类似，不相等运算符在比较的时候也会转换值的数据类型。

**例如**

1 != 2// true   
1 != "1"// false   
1 != '1'// false   
1 != true // false   
0 != false// false

#### 任务

在 if 语句中，添加不相等运算符 !=，这样函数在当 val不等于99 的时候，会返回 "Not Equal"。



### JavaScript 不全等于比较运算符!==

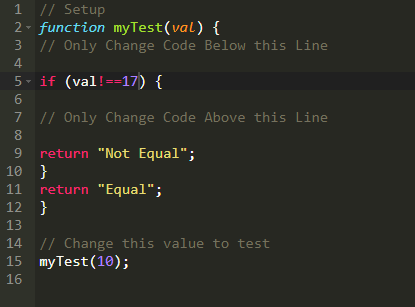
不全等运算符（!==）与全等运算符是相反的。这意味着“不全等”并返回 false 的地方，用全等运算会返回 true，反之亦然。全等运算符不会转换值的数据类型。

**例如**

3 !== 3 // false  
3 !== '3' // true  
4 !== 3 // true

#### 任务

在 if 语句中，添加 不全等运算符，这样如果 val 不全等于 17 的时候，函数会返回 "Not Equal"。



### JavaScript 大于比较运算符>

使用大于运算符（>）来比较两个数字。如果大于运算符左边的数字大于右边的数字，将会返回 true。否则，它返回 false。

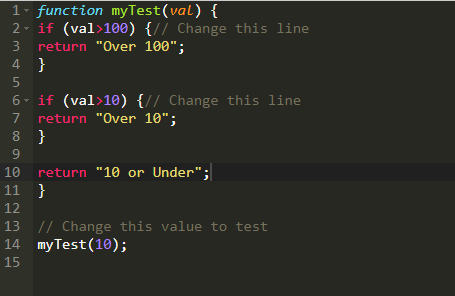
与相等运算符一样，大于运算符在比较的时候，会转换值的数据类型。

#### 举个栗子

5 > 3 // true  
7 > '3' // true  
2 > 3 // false  
'1' > 9// false

#### 任务

添加 大于 运算符到指定的行，使得返回的语句是有意义的。



### JavaScript 大于等于比较运算符>=

使用 大于等于 运算符（>=）来比较两个数字的大小。如果大于等于运算符左边的数字比右边的数字大或者相等，它会返回 true。否则，它会返回 false。

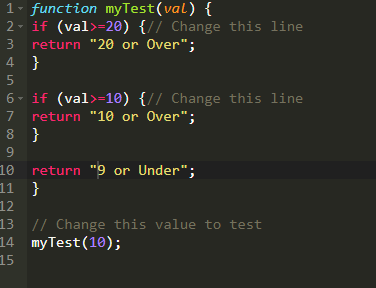
与相等运算符相似，大于等于 运算符在比较的时候会转换值的数据类型。

#### 举个栗子

6>=6// true  
7>= '3' // true  
2>=3// false  
'7' >=9// false

#### 任务

添加 大于等于 运算符到指定行，使得函数的返回语句有意义。



### JavaScript 小于比较运算符<

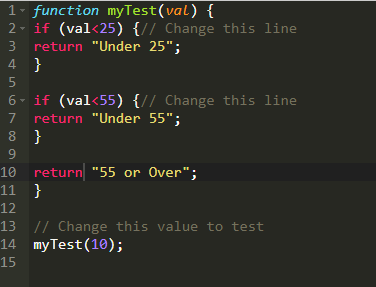
使用 小于 运算符（<）比较两个数字的大小。如果小于运算符左边的数字比右边的数字小，它会返回 true。否则，他会返回 false。与相等运算符类似，小于 运算符在做比较的时候会转换值的数据类型。

#### 举个栗子

2 < 5// true  
'3' < 7// true  
5 < 5// false  
3 < 2// false  
'8' < 4// false

#### 任务

添加 小于 运算符到指定行，使得函数的返回语句有意义。



### JavaScript 小于等于比较运算符<=

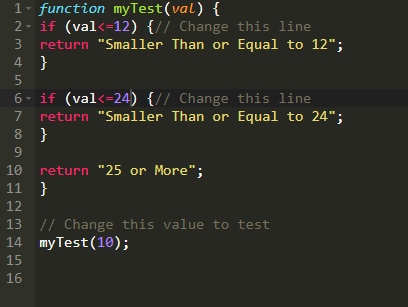
使用 小于等于 运算符（<=）比较两个数字的大小。如果在小于等于运算符，左边的数字小于或者等于右边的数字，它会返回 true。如果在小于等于运算符，左边的数字大于或者等于右边的数字，它会返回 false。与相等运算符类型，小于等于 运算符会转换数据类型。

#### 举个栗子

4 <= 5// true  
'7' <= 7// true  
5 <= 5// true  
3 <= 2// false  
'8' <= 4// false

#### 任务

添加 小于等于 运算符到指定行，使得函数的返回语句有意义。



### JavaScript 使用逻辑与运算符&&比较数值

有时你需要在一次判断中做多个操作。运算符的左边和右边结果必须都是 true，逻辑与 运算符（&&）才会返回 true。

同样的效果可以通过if语句的嵌套来实现：

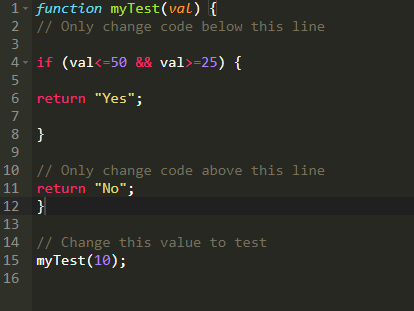
if (num > 5) {  
if (num < 10) {  
return "Yes";  
}  
}  
return "No";

只有当 num 的值在6和9之间（包括6和9）才会返回 "Yes"。相同的逻辑可被写为：

if (num > 5 && num < 10) {  
return "Yes";  
}  
return "No";

#### 任务

结合两个if语句为一个语句，如果 val 小于或等于 50 并且大于或等于 25，返回 "Yes"。否则，将返回 "No"。



### JavaScript 使用逻辑或运算符||比较数值

逻辑或运算符左右两边任何一个操作数是true，逻辑或 运算符 (||) 返回 true。反之，返回 false。

#### 举个栗子

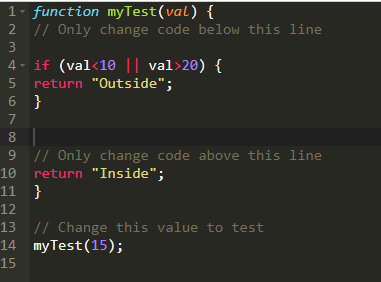
if (num > 10) {  
return "No";  
}  
if (num < 5) {  
return "No";  
}  
return "Yes";

只有当num大于等于5或小于等于10时，函数返回"Yes"。相同的逻辑可以简写成：

if (num > 10 || num < 5) {  
return "No";  
}  
return "Yes";

#### 任务

结合两个if语句为一个语句，如果val不在10和20之间(包括10和20)，返回 "Outside"。反之，返回 "Inside"。



### JavaScript else语句

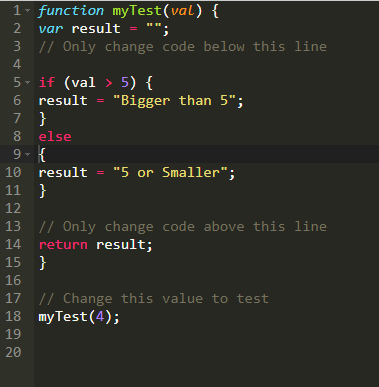
当if语句的条件为真，大括号里的代码执行，那如果条件为假呢？

这时候需要写一个else语句，当条件为假时执行相应的代码。

if (num > 10) {  
return "Bigger than 10";  
} else {  
return "10 or Less";  
}

#### 任务

结合多个if语句为一个if/else语句。



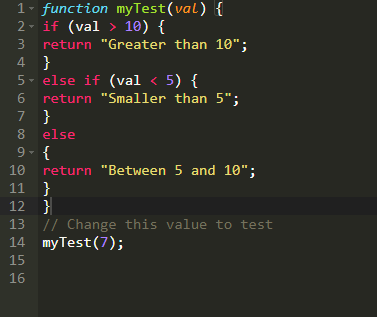
### JavaScript else if语句

如果你有多个条件语句，你可以通过else if语句把 if语句链起来。

if (num > 15) {  
return "Bigger than 15";  
} else if (num < 5) {  
return "Smaller than 5";  
} else {  
return "Between 5 and 15";  
}

#### 任务

使用else if实现同样的效果。



### JavaScript if、else if语句中代码的执行顺序

if、else if语句中代码的执行顺序是很重要的。

在条件判断语句中，代码的执行顺序自上而下，所以必须区分清除代码的执行顺序。

这有两个例子。

第一个例子：

function foo(x) {  
if (x < 1) {  
return "Less than one";  
} else if (x < 2) {  
return "Less than two";  
} else {  
return "Greater than or equal to two";  
}  
}

第二个例子更改了代码的执行顺序：

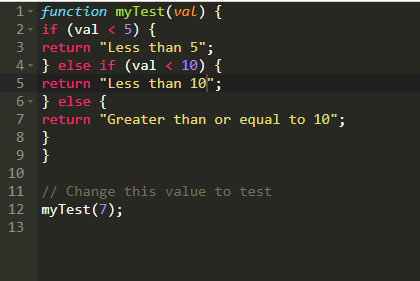
function bar(x) {  
if (x < 2) {  
return "Less than two";  
} else if (x < 1) {  
return "Less than one";  
} else {  
return "Greater than or equal to two";  
}  
}

这两个函数看起来相似，实则大不相同。

foo(0) // "Less than one"  
bar(0) // "Less than two"

#### 任务

更改函数的逻辑顺序以便通过所有的测试用例。



### JavaScript 同时使用if、else if 语句

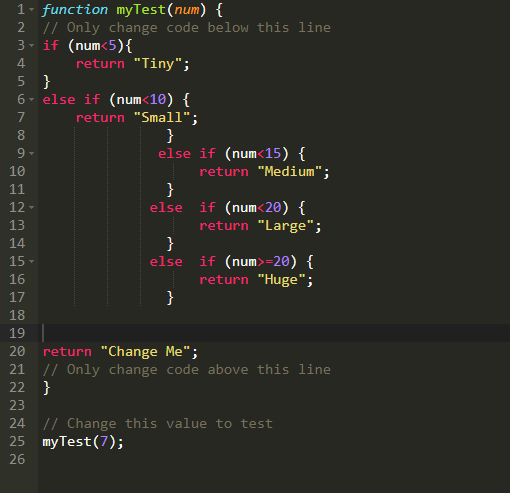
if/else 语句串联在一起可以实现复杂的逻辑，这是多个if/else if 语句串联在一起的伪代码：

if (condition1) {  
statement1  
} else if (condition2) {  
statement2  
} else if (condition3) {  
statement3  
. . .  
} else {  
statementN  
}

#### 任务

把if/else if语句串联起来实现下面的逻辑：

num < 5 - return "Tiny"  
num <10 - return "Small"  
num < 15 - return "Medium"  
num < 20 - return "Large"  
num >= 20- return "Huge"



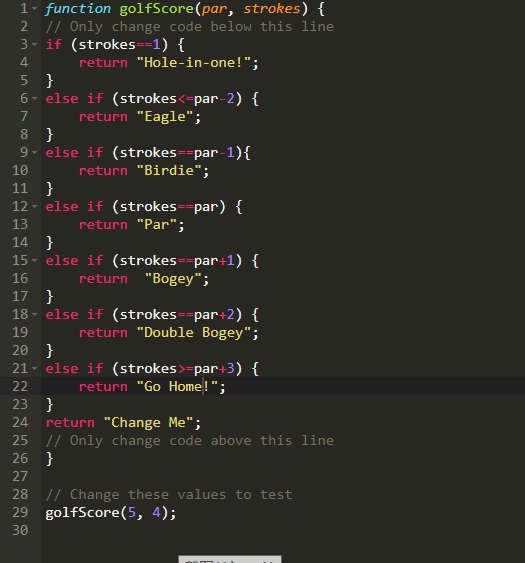
### Javascript 逻辑运算综合实战

在高尔夫golf游戏中，每个洞都有自己的标准杆数par，代表着距离。根据你把球打进洞所挥杆的次数strokes，可以计算出你的高尔夫水平。

函数将会传送2个参数，分别是标准杆数par 和 挥杆次数strokes ，根据下面的表格返回正确的水平段位。

| **Strokes** | **Return** |
| --- | --- |
| 1 | "Hole-in-one!" |
| <= par - 2 | "Eagle" |
| par - 1 | "Birdie" |
| par | "Par" |
| par + 1 | "Bogey" |
| par + 2 | "Double Bogey" |
| >= par + 3 | "Go Home!" |

par 和 strokes 必须是数字而且是正数。



### JavaScript 使用switch语句进行多选项选择

如果你有非常多的选项需要选择，可以使用switch语句。根据不同的参数值会匹配上不同的case分支，语句会从第一个匹配的case分支开始执行，直到碰到break就结束。

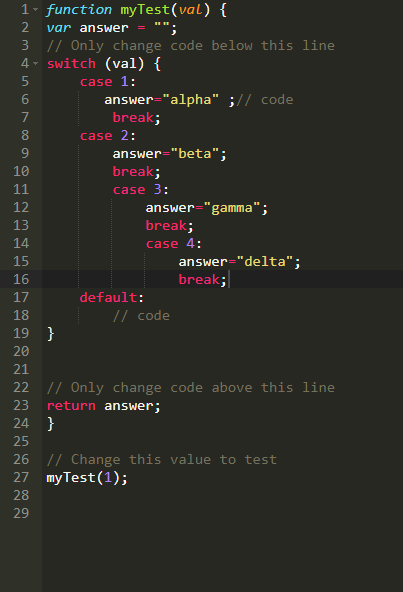
这是一个伪代码案例：

switch (num) {  
case value1:  
statement1;  
break;  
case value2:  
statement2;  
break;  
...  
case valueN:  
statementN;  
break;  
}

测试case 值使用严格等于，break关键字告诉javascript停止执行语句。如果没有break关键字，下一个语句会继续执行。

#### 任务

写一个测试 val的switch语句，并且根据下面的条件来设置不同的answer：  
1 - "alpha"  
2 - "beta"  
3 - "gamma"  
4 - "delta"



### JavaScript 在switch语句中添加default语句

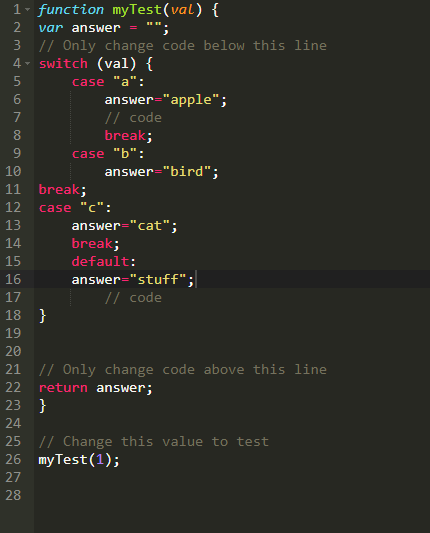
在switch 语句中你可能无法用case来指定所有情况，这时你可以添加default语句。当再也找不到case匹配的时候default语句会执行，非常类似于if/else组合中的else语句。

default语句应该是最后一个case。

switch (num) {  
case value1:  
statement1;  
break;  
case value2:  
statement2;  
break;  
...  
default:  
defaultStatement;  
}

#### 任务

写一个根据下面的条件来设置answer的switch语句：  
"a" - "apple"  
"b" - "bird"  
"c" - "cat"  
default - "stuff"



### JavaScript switch语句中的多个相同选项判断

如果switch语句中的case分支的break 语句漏掉了，后面的 case语句会一直执行直到遇到break。如果你有多个输入值和输出值一样，可以试试下面的switch语句：

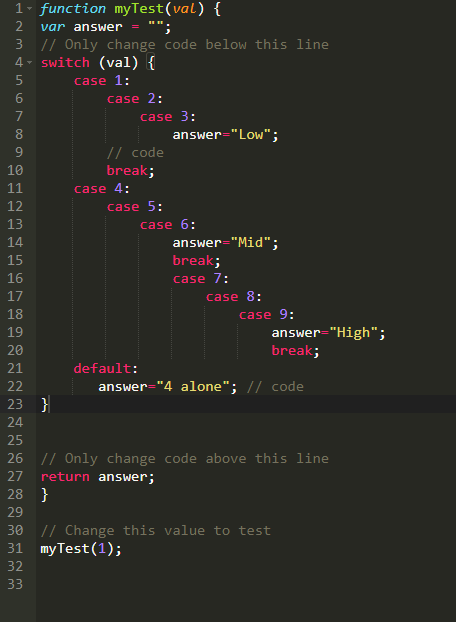
switch(val) {  
case 1:  
case 2:  
case 3:  
result = "1, 2, or 3";  
break;  
case 4:  
result = "4 alone";  
}

分支1、2、3将会产生相同的输出结果。

#### 任务

写一个根据下面的范围来设置answer 的switch语句：  
1-3 - "Low"  
4-6 - "Mid"  
7-9 - "High"

**提示**  
你需要为每一个包含数字的范围准备一个answer语句。



### JavaScript 使用switch语句替换串联的if、else if语句

如果你有多个选项需要选择，switch 语句写起来会比多个串联的if/if else语句容易些，譬如:

if (val === 1) {  
answer = "a";  
} else if (val === 2) {  
answer = "b";  
} else {  
answer = "c";  
}

可以被下面替代：

switch (val) {  
case 1:  
answer = "a";  
break;  
case 2:  
answer = "b";  
break;  
default:  
answer = "c";  
}

#### 任务

把串联的 if/if else 语句改成 switch 语句。



### JavaScript 直接在函数中返回boolean值

所有的比较操作符返回的都是一个boolean值，要么是 true 要么是false 。

使用 if/else语句来做比较然后返回true或false已经成为大家的共识:

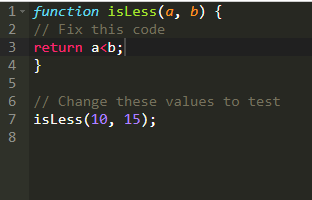
function isEqual(a,b) {  
if (a === b) {  
return true;  
} else {  
return false;  
}  
}

因为=== 总是返回 true 或 false，所以我们可以直接返回比较的结果：

function isEqual(a,b) {  
return a === b;  
}

#### 任务

移除isLess函数的if/else语句但不影响函数的功能。



当a<b时，返回ture ,否则，返回 false.

### JavaScript 在函数中使用return跳出函数

当代码执行到return语句时，函数返回一个结果就结束运行了，return后面的语句根本不会执行。

#### 举个栗子

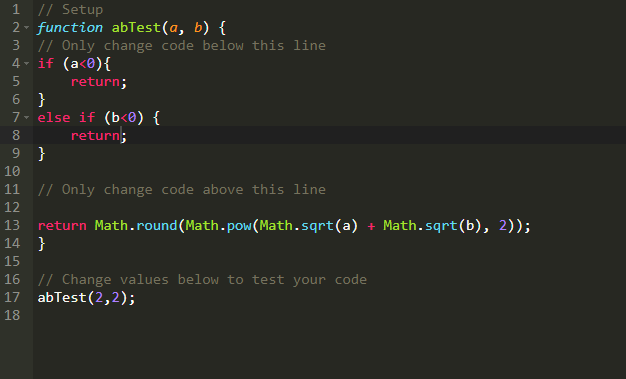
function myFun() {  
console.log("Hello");  
return "World";  
console.log("byebye")  
}  
myFun();

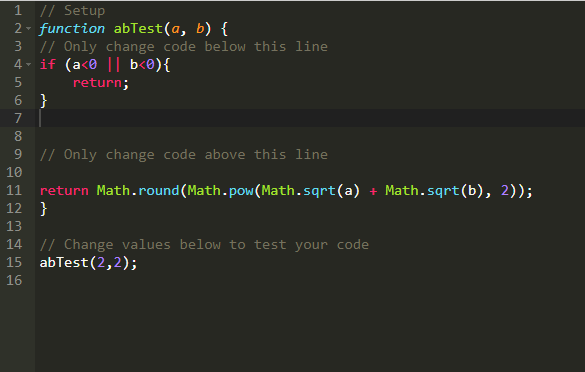
上面的代码输出"Hello"到控制台、返回 "World"，但没有输出"byebye"，因为函数遇到return语句就退出了。

#### 任务

修改函数abTest当a或b小于0时，函数立即返回一个undefined并退出。

**提示**  
记住[undefined](https://www.w3cschool.cn/codecamp/understanding-uninitialized-variables)，是一个关键字，而不是一个字符串。





### JavaScript 条件判断算法综合实战

在赌场21点游戏中，玩家可以通过计算牌桌上已经发放的卡牌的高低值来让自己在游戏中保持优势，这就叫[21点算法](https://www.douban.com/note/273781969/)。

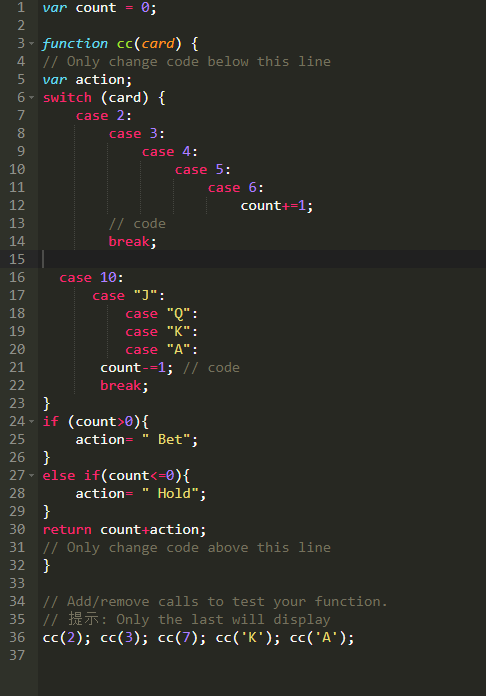
根据下面的表格，每张卡牌都分配了一个值。如果卡牌的值大于0，那么玩家应该追加赌注。反之，追加少许赌注甚至不追加赌注。

| **Count Change** | **Cards** |
| --- | --- |
| +1 | 2, 3, 4, 5, 6 |
| 0 | 7, 8, 9 |
| -1 | 10, 'J', 'Q', 'K','A' |

你需要写一个函数，实现21点算法，它根据参数 card的值来递增或递减变量count，函数返回一个由当前count和 "Bet"(count>0)或"Hold"(count<=0) 拼接的字符串。注意count和"Bet" 或 "Hold"应该用空格分开。

**例如：**  
"-3 Hold"  
"5 Bet"

**提示**  
既然card的值为7、8、9时，count值不变，那我们就可以忽略这种情况。



### JavaScript 对象操作

你之前可能听说过对象 object 。

对象和数组很相似，数组是通过索引来访问和修改数据，对象是通过属性来访问和修改数据的。

这是一个示例对象：

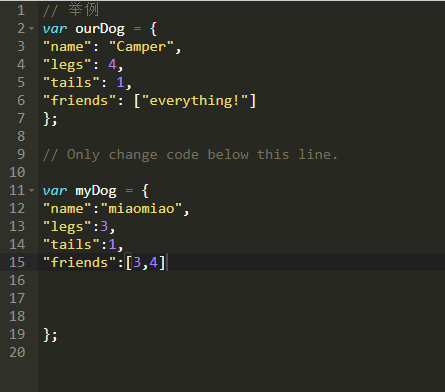
var cat = {  
"name": "Whiskers",  
"legs": 4,  
"tails": 1,  
"enemies": ["Water", "Dogs"]  
};

对象适合用来存储结构化数据，就和真实世界的对象一模一样，比如一只猫。

#### 任务

创建一个叫做myDog的对象，它里面有这些属性： 名称"name"、 腿"legs", 尾巴"tails"、朋友"friends"。

你可以设置对象属性为任何你想要的值，只要 "name" 是字符串、 "legs" 和 "tails" 是数字、"friends" 是数组。



### avaScript 使用点操作符.读取对象属性

有两种方式访问对象属性，一个是点操作符(.)，一个是中括号操作符([])。

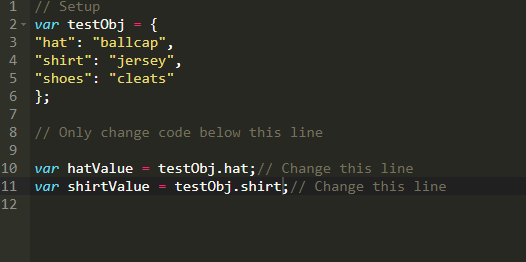
当你知道属性的名称的时候，使用点操作符。

这是一个使用点操作符读取对象属性的例子：

var myObj = {  
prop1: "val1",  
prop2: "val2"  
};  
var prop1val = myObj.prop1; // val1  
var prop2val = myObj.prop2; // val2

#### 任务

通过点操作符读取对象testObj，把hat的属性值赋给变量hatValue，把shirt的属性值赋给shirtValue。



### JavaScript 使用[]读取对象属性

第二种访问对象的方式就是中括号操作符([])，如果你想访问的属性的名称有一个空格，这时你只能使用中括号操作符([])。

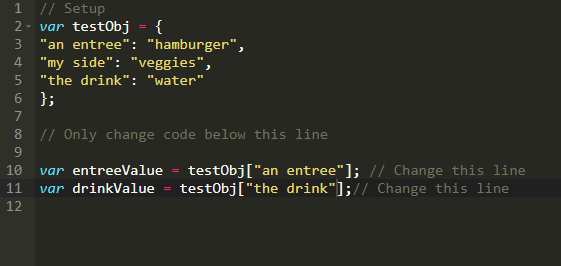
这是一个使用中括号操作符([])读取对象属性的例子：

var myObj = {  
"Space Name": "Kirk",  
"More Space": "Spock"  
};  
myObj["Space Name"]; // Kirk  
myObj['More Space']; // Spock

提示：属性名称中如果有空格，必须把属性名称用单引号或双引号包裹起来。

#### 任务

用中括号操作符读取对象testObj的属性"an entree"值和属性"the drink"值。



### JavaScript 使用变量访问对象属性

中括号操作符的另一个使用方式是用变量来访问一个属性。当你需要遍历对象的属性列表或查表时，这种方式极为有用。

这有一个使用变量来访问属性的例子：

var someProp = "propName";  
var myObj = {  
propName: "Some Value"  
}  
myObj[someProp]; // "Some Value"

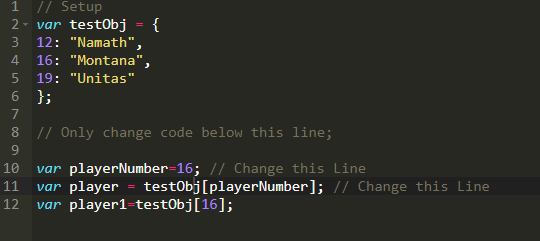
还有更多：

var myDog = "Hunter";  
var dogs = {  
Fido: "Mutt", Hunter: "Doberman", Snoopie: "Beagle"  
}  
var breed = dogs[myDog]; // "Hunter"  
console.log(breed)// "Doberman"

提示：当我们通过变量名访问属性的时候，不需要给变量名包裹引号。因为实际上我们使用的是变量的值，而不是变量的名称。

#### 任务

使用变量playerNumber，通过中括号操作符找到testObj中的16。



### JavaScript 更新对象属性

当你创建了一个对象后，你可以用点操作符或中括号操作符来更新对象的属性。

举个例子，让我们看看 ourDog:

var ourDog = {  
"name": "Camper",  
"legs": 4,  
"tails": 1,  
"friends": ["everything!"]  
};

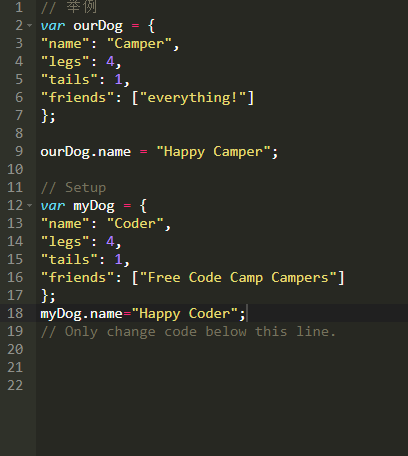
让我们更改它的名称为 "Happy Camper"，这有两种方式来更新对象的name属性：

ourDog.name = "Happy Camper";

ourDog["name"] = "Happy Camper";

#### 任务

更新 myDog 对象的name属性，让它的名字从 "Coder" 变成 "Happy Coder"。



### JavaScript 给对象添加属性

你也可以像更改属性一样给对象添加属性。

看看我们是如何给ourDog添加 "bark"属性：

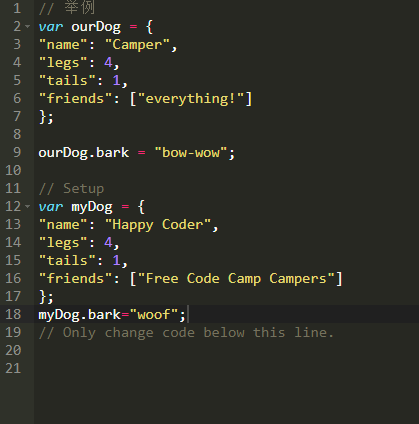
ourDog.bark = "bow-wow";

或者

ourDog["bark"] = "bow-wow";

#### 任务

给myDog添加一个"bark" 属性，设置它的值为狗的声音，例如："woof"。



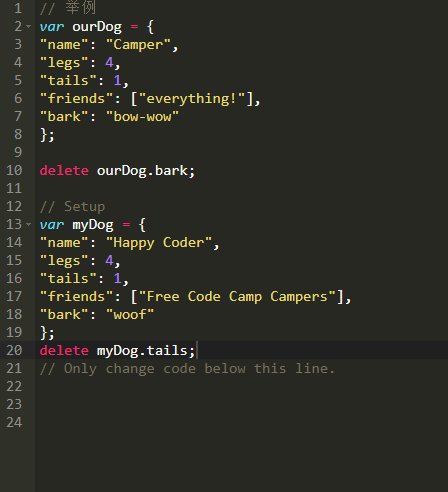
### JavaScript 删除对象属性

我们同样可以删除对象的属性，例如：

delete ourDog.bark;

#### 任务

删除myDog对象的"tails"属性。



### JavaScript 使用对象进行查找值

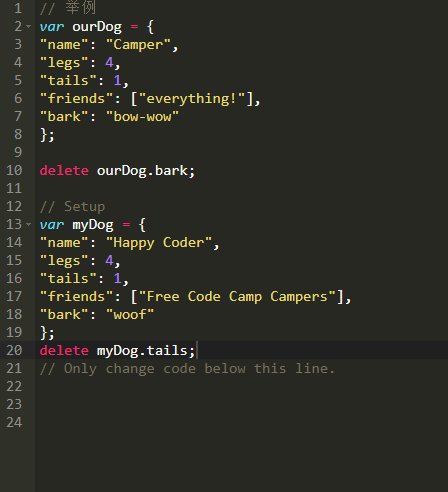
对象和字典一样，可以用来存储键/值对。如果你的数据跟对象一样，你可以用对象来查找你想要的值，而不是使用switch或if/else语句。当你知道你的输入数据在某个范围时，这种查找方式极为有效。

这是简单的反向字母表：

var alpha = {  
1:"Z",  
2:"Y",  
3:"X",  
4:"W",  
...  
24:"C",  
25:"B",  
26:"A"  
};  
alpha[2]; // "Y"  
alpha[24]; // "C"  
  
var value = 2;  
alpha[value]; // "Y"

#### 任务

把switch语句转化为一个叫做lookup的对象。



### JavaScript 检查对象属性

有时检查一个对象属性是否存在是非常有用的，我们可以用.hasOwnProperty(propname)方法来检查对象是否有该属性。如果有返回true，反之返回 false。

**举例**

var myObj = {  
top: "hat",  
bottom: "pants"  
};  
myObj.hasOwnProperty("top");// true  
myObj.hasOwnProperty("middle"); // false

#### 任务

修改函数checkObj检查myObj是否有checkProp属性，如果属性存在，返回属性对应的值，如果不存在，返回 "Not Found"。

注意：如果你需要通过变量来访问对象的属性值，请用中括号操作符，点操作符不支持变量。



### JavaScript JSON操作

JavaScript Object Notation 简称 JSON，它使用JavaScript对象的格式来存储数据。JSON是灵活的，因为它允许 数据结构 是 字符串，数字，布尔值，字符串，和 对象 的任意组合。

这里是一个JSON对象的示例：

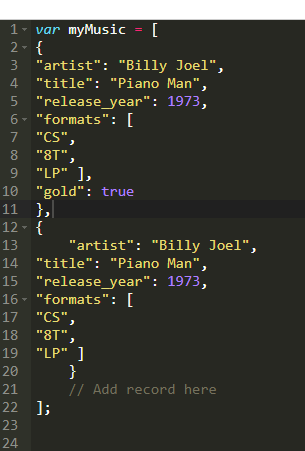
var ourMusic = [  
{  
"artist": "Daft Punk",  
"title": "Homework",  
"release\_year": 1997,  
"formats": [   
"CD",   
"Cassette",   
"LP" ],  
"gold": true  
}  
];

这是一个对象数组，并且对象有各种关于专辑的 详细信息。它也有一个嵌套的 formarts 的数组。附加专辑记录可以被添加到数组的最上层。

**提示**  
数组中有多个 JSON 对象的时候，对象与对象之间要用逗号隔开。

#### 任务

添加一个新专辑到 myMusic 的JSON对象。添加 artist 和 title 字符串，release\_year 数字和 formats 字符串数组。



### JavaScript 获取JSON属性值

通过串联起来的点操作符或中括号操作符来访问JSON对象的嵌套属性。

下面是一个嵌套的JSON对象：

var ourStorage = {  
"desk": {  
"drawer": "stapler"  
},  
"cabinet": {  
"top drawer": {   
"folder1": "a file",  
"folder2": "secrets"  
},  
"bottom drawer": "soda"  
}  
}  
ourStorage.cabinet["top drawer"].folder2;// "secrets"  
ourStorage.desk.drawer; // "stapler"

#### 任务

检索JSON对象 myStorage 中嵌套属性 glove box 的值。因为属性的名字带有空格，请使用中括号操作符来访问属性的值。



### JavaScript 获取JSON数组值

正如我们在前面的例子所见，JSON对象可以嵌套对象和数组。与访问嵌套对象一样，用中括号操作符同样可以访问嵌套数组。

下面是如何访问嵌套数组的例子：

var ourPets = {   
"cats": [  
"Meowzer",  
"Fluffy",  
"Kit-Cat"  
],  
"dogs": [  
"Spot",  
"Bowser",  
"Frankie"  
]  
};  
ourPets.cats[1]; // "Fluffy"  
ourPets.dogs[0]; // "Spot"

#### 任务

使用点操作符和中括号操作符来检索变量 myPlants 的第二棵树。



### JavaScript JSON集合操作

右边有一个JSON对象，代表着你的专辑集。每一张专辑由一个唯一的id标识，并具有多种属性。但并非所有的专辑都有完整的信息。

写一个函数，它有个三个参数，id、prop、 value。

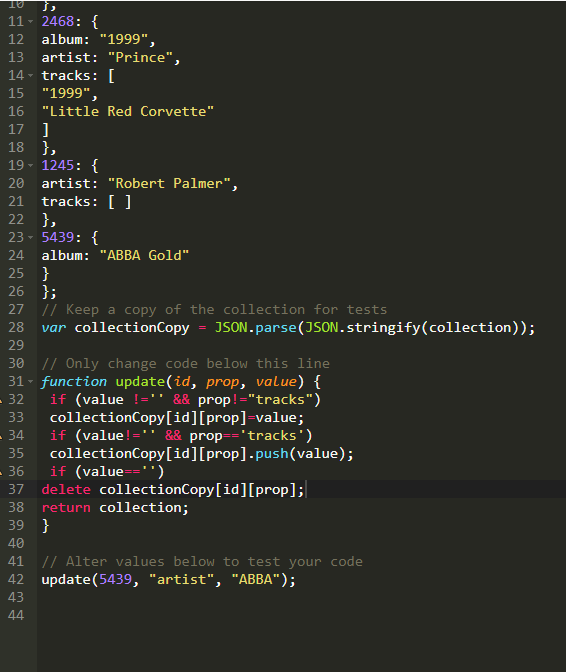
如果 value !='' 而且prop != 'tracks' ，collectionCopy[id][prop]=value;。

如果 value !='' 而且prop == 'tracks' ，collectionCopy[id][prop].push(value);。

如果 value == '' ，delete collectionCopy[id][prop];。

谨记：函数返回的永远是整个对象。

**提示**  
使用中括号操作符来 [访问对象的变量属性](https://www.w3cschool.cn/codecamp/accessing-objects-properties-with-variables.html)。



### JavaScript 使用for语句循环迭代

一个条件语句只能执行一次代码，而通过一个循环语句你可以多次执行代码。

JavaScript 中最常见的循环就是“for循环”因为它可以允许特定次数的循环。

for循环中的三个表达式用分号隔开：

for ([初始化]; [条件判断]; [计数器])

初始化语句只会在执行循环开始之前执行一次。它通常用于定义和设置你的循环变量。

条件判断语句会在每一轮循环的开始执行，只要条件判断为 true 就会继续执行循环。当条件为 false的时候，循环将停止执行。这意味着，如果条件在一开始就为 false，这个循环将不会执行。

计数器是在每一轮循环结束时执行，通常用于递增或递减。

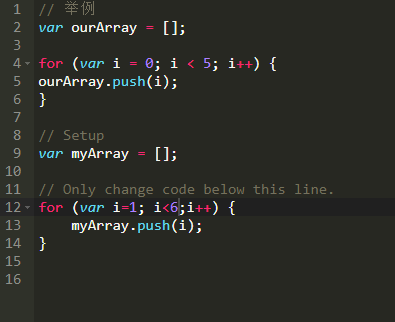
在下面的例子中，先初始化i = 0，条件 i < 5 为真，进入第一次循环，执行大括号里的代码，第一次循环结束。递增i的值，条件判断，就这样依次执行下去，直到条件判断为假，整个循环结束。

var ourArray = [];  
for (var i = 0; i < 5; i++) {  
ourArray.push(i);  
}

最终 ourArray 的值为 [0,1,2,3,4].

#### 任务

使用 for 循环把从 1 到 5 添加进 myArray 中。



### JavaScript 使用for语句循环按奇数顺序迭代

for循环不必按照顺序来迭代，通过更改我们的 表达式，我们可以按照偶数顺序来迭代。

初始化 i = 0，当 i < 10 的时候继续循环。

i += 2 让 i 每次循环之后增加2。

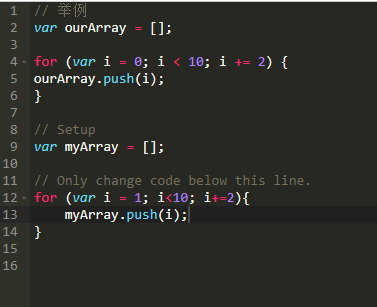
var ourArray = [];  
for (var i = 0; i < 10; i += 2) {  
ourArray.push(i);  
}

循环结束后，ourArray 的值为 [0,2,4,6,8]。

让我们改变 表达式，这样我们可以用奇数来数。

### 任务

写一个 for 循环，把从1到9的奇数添加到 myArray。



### JavaScript 使用for循环逆向迭代

for循环也可以逆向迭代，只要我们定义好合适的条件。

为了能够从后往前两两倒数，我们需要改变我们的 初始化，条件判断和 计数表达式。

我们让 i = 10，并且当 i > 0 的时候才继续循环。我们使用 i-=2来让 i 每次循环递减 2。

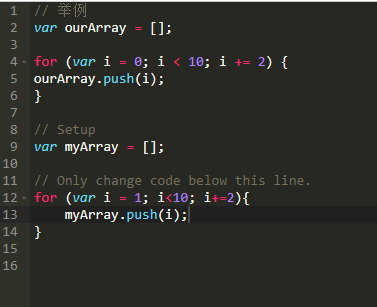
var ourArray = [];  
for (var i=10; i > 0; i-=2) {  
ourArray.push(i);  
}

循环结束后，ourArray 的值为 [10,8,6,4,2]。

让我们改变 初始化 和 计数表达式，这样我们就可以按照奇数从后往前两两倒着数。

#### 任务

使用一个 for 循环，把9到1的奇数添加进 myArray。



### JavaScript 使用for循环迭代输出数组

JavaScript中的常见任务是迭代数组的内容，一种方法是使用for循环。

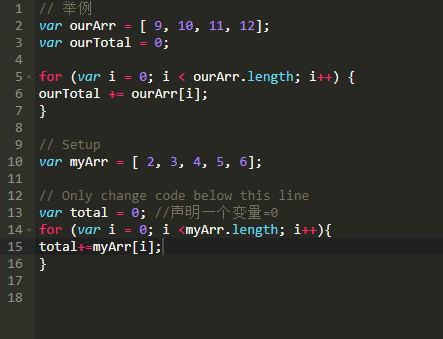
下面的代码将输出数组arr 的每个元素到控制台：

var arr = [10,9,8,7,6];  
for (var i=0; i < arr.length; i++) {  
console.log(arr[i]);  
}

记住数组的索引从零开始的，这意味着数组的最后一个元素的下标是：数组的长度 - 1。我们这个循环的 条件 是 i < arr.length，当 i 的值为 长度-1 的时候循环就停止了。

#### 任务

声明并初始化一个变量 total 为 0。使用 for 循环，使得 total的值为 myArr 的数组中的每个元素的值的总和。



### JavaScript 循环语句综合实战

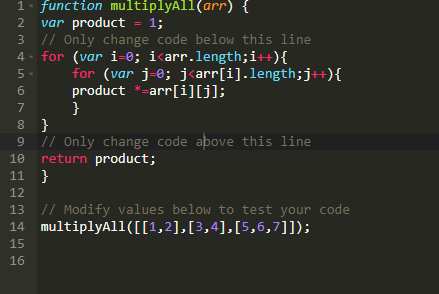
如果你有一个二维数组，可以使用嵌套循环的逻辑，先遍历外面的数组，再遍历里面的子数组。下面是一个例子：

var arr = [  
[1,2], [3,4], [5,6]  
];  
for (var i=0; i < arr.length; i++) {  
for (var j=0; j < arr[i].length; j++) {  
console.log(arr[i][j]);  
}  
}

这样就可以一次输出 arr 中的每个子元素。请注意，对于内部循环，我们可以通过 arr[i] 的 .length 来获得子数组的长度，因为 arr[i] 的本身就是一个数组。

#### 任务

修改函数 multiplyAll，以获得 arr 内部数组的每个数字相乘的结果 product。



### JavaScript 使用while语句循环迭代

您可以使用循环多次运行相同的代码；

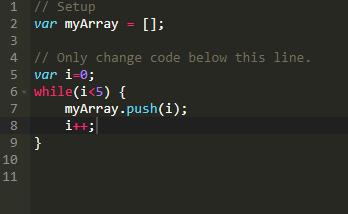
另一种类型的 JavaScript 循环被称为while循环，因为它运行“while”一个指定的条件为真，一旦该条件不再为真，就停止

var ourArray = [];   
var i = 0;   
while(i < 5) {   
ourArray.push(i);   
i++;   
}

让我们尝试通过将值推送到数组来获得一个while循环

#### 任务

通过一个 while 循环，把从 0 到 4 的值添加到 myArray 中。



### JavaScript 使用循环语句查找通讯录

我们有一个联系人列表，里面存储着不同联系人的数组对象。

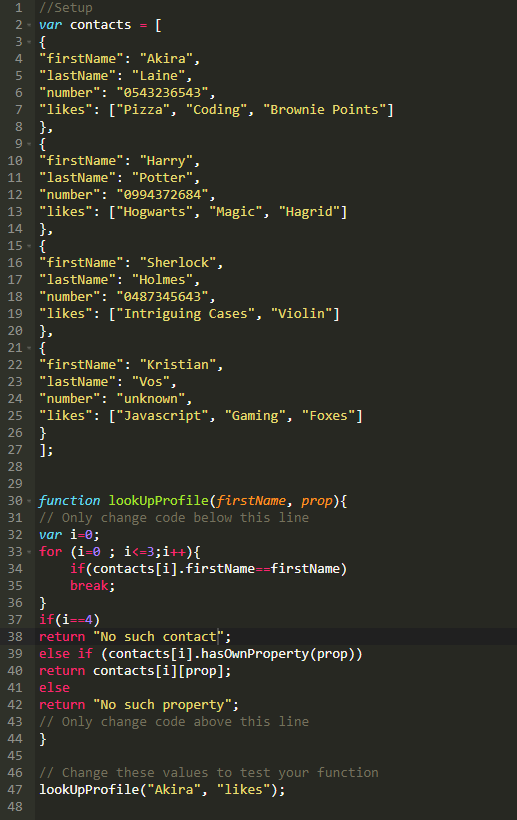
函数 lookUpProfile 有两个预定义参数：firstName值和prop属性 。

该函数应检查firstName是实际联系人的firstName，给定的属性（prop）是该联系人的属性。

如果它们都存在，函数返回prop属性对应的值。

如果firstName 值不存在，返回 "No such contact"。

如果prop 属性不存在，返回 "No such property"。



### JavaScript 使用random()生成随机小数

随机数对于创建随机行为很有用,计算机的行为只有两种：确定性和随机性。

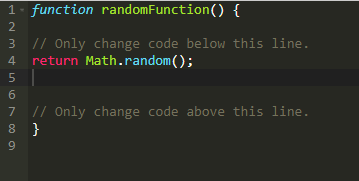
而随机数最适合用来创建这种随机行为。

Math.random()用来生成一个在0(包括0)到1(不包括1)之间的随机小数，因此Math.random()可能返回0但绝不会返回1。

**提示**  
随后的函数都会在return执行前调用，所以我们可以直接返回Math.random()的值。

#### 任务

更改randomFunction 来生成一个随机数取代 0。



### JavaScript 使用random()生成随机数

很好，我们可以生成随机的小数，但如果我们使用它来生成随机的整数，这将是更有用的机整数。

1. 用 Math.random() 生成一个随机小数。
2. 把这个随机小数乘以 20。
3. 用 Math.floor() 向下取整 获得它最近的整数。

记住 Math.random() 永远不会返回 1。同时因为我们是在用 Math.floor() 向下取整，所以最终我们获得的结果不可能有 20。这确保了我们获得了一个在0到19之间的整数。

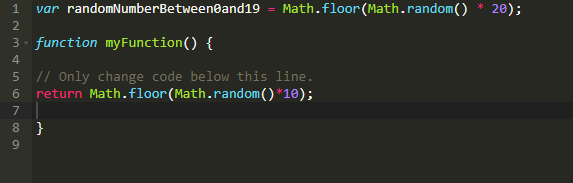
把操作连缀起来，代码类似于下面：

Math.floor(Math.random() \* 20);

我们先调用 Math.random()，把它的结果乘以20，然后把上一步的结果传给 Math.floor()，最终通过向下取整获得最近的整数。

#### 任务

生成一个 0 到 9之间的随机整数。



### JavaScript 使用random()在一个范围内生成随机数

我们之前生成的随机数是在0到某个数之间，现在我们可以生成一个落在两个特定数字范围内的随机数间。

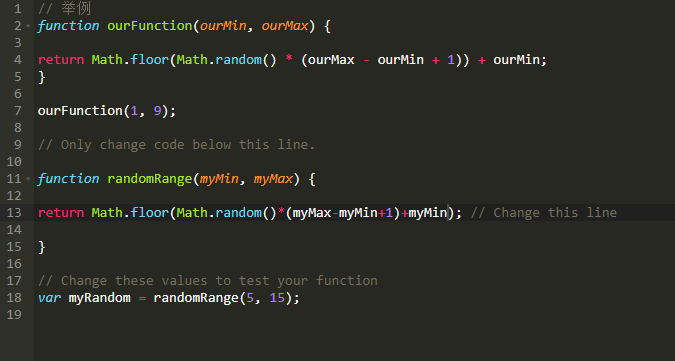
我们需要定义一个最小值和一个最大值。

下面是我们将要使用的方法，仔细看看并尝试理解这行代码正在干什么：

Math.floor(Math.random() \* (max - min + 1)) + min

#### 任务

创建一个叫randomRange的函数，参数为myMin和myMax，返回一个在myMin(包括myMin)和myMax(包括myMax)之间的随机数。



### JavaScript 使用正则表达式操作字符串

Regular expressions 正则表达式被用来根据某种匹配模式来寻找strings中的某些单词。

例如：如果我们想要找到字符串The dog chased the cat中单词 the，我们可以使用下面的正则表达式: /the/gi

我们可以把这个正则表达式分成几段：

/ 是这个正则表达式的头部

the 是我们想要匹配的模式

/ 是这个正则表达式的尾部

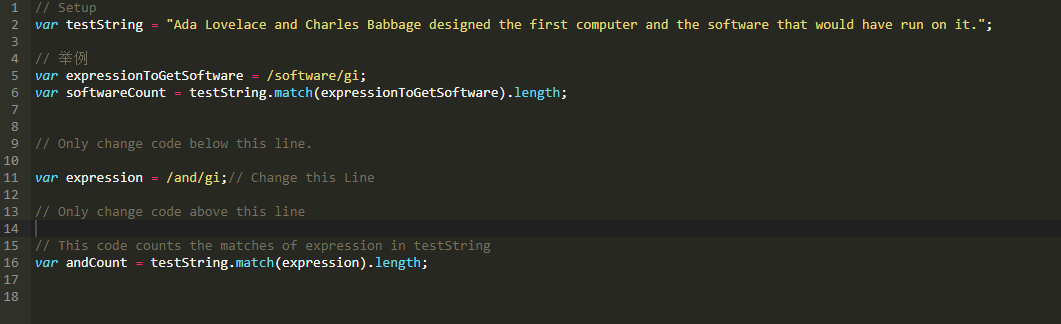
g 代表着 global(全局)，意味着返回所有的匹配而不仅仅是第一个。

i 代表着忽略大小写，意思是当我们寻找匹配的字符串的时候忽略掉字母的大小写。

#### 任务

用全局、忽略大小写的模式选取字符串 testString中所有的单词 and。

你可以尝试把 . 替换成 and。



### JavaScript 使用正则表达式选取数值

我们可以在正则表达式中使用特殊选择器来选取特殊类型的值。

特殊选择器中的一种就是数字选择器\d，意思是被用来获取一个字符串的数字。

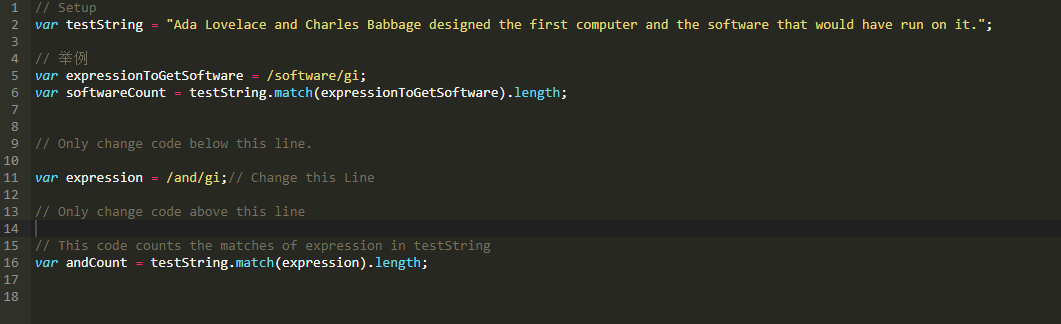
在JavaScript中, 数字选择器类似于: /\d/g。

在选择器后面添加一个加号标记(+)，例如：/\d+/g，它允许这个正则表达式匹配一个或更多数字。

尾部的g是'global'的简写，意思是允许这个正则表达式 找到所有的匹配而不是仅仅找到第一个匹配。

#### 任务

用 \d 选择器来选取字符串中的所有数字，允许一个或多个数字的可能性。



### JavaScript 使用正则表达式选取空白字符

我们也可以使用正则表达式选择器 \s 来查找一个字符串中的空白。

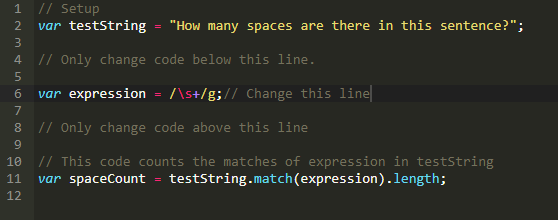
空白字符有 " " (空格符)、\r (回车符)、\n (换行符)、\t (制表符) 和 \f (换页符)。

空白正则表达式类似于：

/\s+/g

#### 任务

用 \s 选取句子中的所有空白字符。



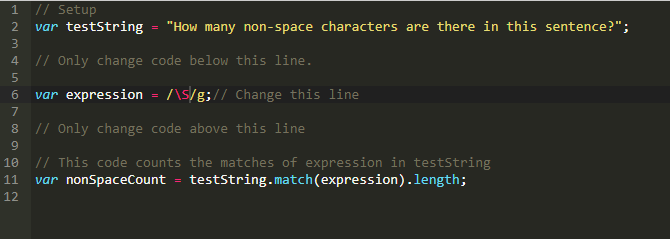
### JavaScript 使用正则表达式反转匹配

你可以用正则表达式选择器的大写版本来反转任何匹配（相反意思）。

例如：\s 匹配任何空白字符，\S 匹配任何非空白字符。

#### 任务

用 /\S/g 来匹配字符串testString中的所有非空白字符。



### JavaScript 综合运用开发老虎机游戏

现在把我们之前的所学的知识点结合起来完成一个老虎机游戏。

首先我们生成3个随机数，范围在1到3之间。

分别用 slotOne、slotTwo、slotThree来存储着3个随机数。

用我们之前的所学来生成[随机数](https://www.w3cschool.cn/codecamp/generate-random-whole-numbers-within-a-range))：

Math.floor(Math.random() \* (3 - 1 + 1)) + 1;