Introduction

I find that many people, even those in computer related fields, don't know anything about programming. It's not that they don't want to learn, it's just that they've never taken the time because some of the concepts are rather abstract and conventional books are rather dry.

By using simple language and illustrations, I hope to make the abstract concepts more concrete. I hope that this book will make programming accessible to people who have an interest, but have not taken dozens of computer science classes and don't have the time (or inspiration) to read books full of techno-babble.

For this book, we'll be using the programming language javaScript. Just like we speak English to each other, we'll be "speaking" javaScript to the computer. I chose to teach javaScript because it is more or less universally accessible. Anyone with a computer, a text editor (like Microsoft Notepad), and a web browser (like Internet Explorer) can write simple programs (called 'scripts') in javaScript.

Once you learn the fundamentals, it's relatively easy to learn a new programming language. Just as in speaking or writing, once you learn the basics, it's just a matter of learning the syntax and vocabulary to learn a new language. For right now, don't worry too much about the syntax or vocabulary… you'll pick it up as we go along.

The purpose of this book is not to teach you javaScript. It is to teach you the fundamental concepts of programming, in general.

Setup

Before we can jump right in, we'll need to set up a few things so that you can follow along. We'll need three things, two of which are most likely already on your computer.

First thing: **A Text Editor**. A text editor is a simple program that allows you to edit text and only text. A word processor (like Microsoft Word) will not work because it allows you to edit formatting which insert extra formatting markup that will prevent your code from running properly. Text editors that will work include **Notepad** (if you are using Windows) or **TextEdit** (if you are using a Mac).

Second thing: **A Web Browser.** A web browser is a program that allows you to view websites. In addition to viewing websites online, you can also use this program to view HTML files located on your home computer. Popular web browsers include **Mozilla Firefox**, **Microsoft Internet Explorer**, or **Safari**.

Notepad in Windows XP

Notepad in Windows Vista

TextEdit in Mac OSX

Internet Explorer

Mozilla Firefox

Safari

Third thing: **An HTML file.** This is the file that you'll write the code in. To create the file, open your text editor and save a new file as `beginnerscript.html`. Make sure that your text editor has not added ".txt" onto the end of the filename. If it did, you can just rename the file and remove the extra ".txt".

Copy the code on the next page into your text editor. Save the file and open it in your web browser.

If you did everything right, a text box should pop up reading "Hello World!" and the page should have "An Illustrated Guide to Programming for Absolute Beginners" in big bold letters.

Any code written between `<script type="text/javascript">` and `</script>` will be run by the web browser as soon as the page is loaded. Whenever you change the code, you'll need to save the beginnerscript.html file in the text editor and refresh it in the web browser. The browser will re-run the code every time it is refreshed. In this case the line that reads `alert("Hello World")` is being run and results in a popup box.

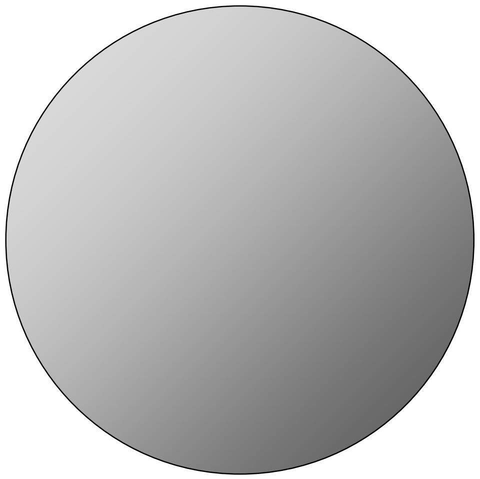
For the rest of this book, a callout bubble will be used to represent the alert box.

```
<html>
      <head>
            <title>My Script</title>
            <script type="text/javascript">
                  alert("Hello World!");
            </script>
      </head>
      <body>
            <h1>
                  An Illustrated Guide to Programming for Absolute Be-
ginners
            </h1>
      </body>
```

```
alert("Hello World");
```

Objects (part 1)

An **object** is any “thing” that you can do stuff to. In the real world, objects would include things cars, mailboxes, and footballs. In programming, objects include things like numbers and words. If you can do something to the thing, it is an object. To help illustrate this, our programming objects will be represented by a real world object: a simple ball.



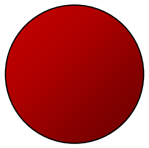
There are two basic types of objects that we will be dealing with.

STRINGS are bits of text. In the code, strings are bits of text with quotation marks (“ “) around them. In the illustrations, strings will be red balls with words on them.

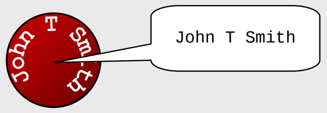
```
alert( “John T Smith” );
```

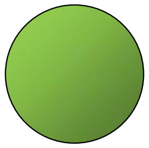
NUMBERS are numbers. In the code, a number is a simply a number with no quotation marks around them. In the illustrations, numbers will be green balls with numbers on them.

```
alert( 123 );
```

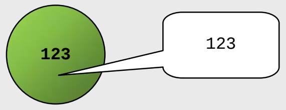


```
alert( “John T Smith” );
```





```
alert( 123 );
```



These basic objects can be added together with the "+" operator.

Adding numbers will result in the sum of the numbers.

```
alert( 2 + 2 + 2 );
```

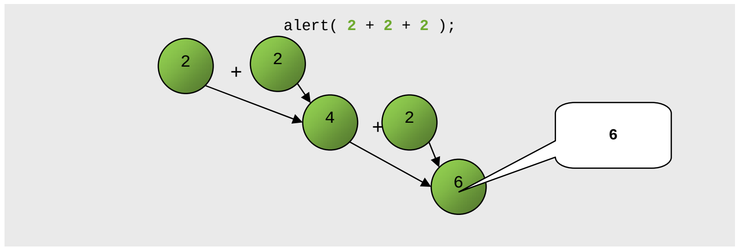
Subtraction works the same way.

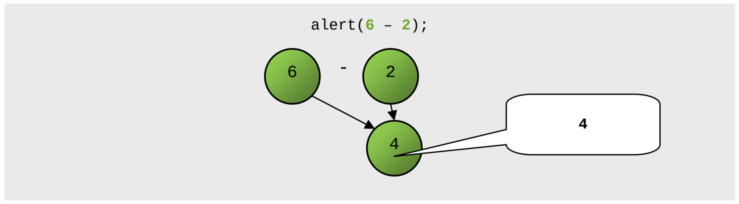
```
alert( 6 – 2);
```

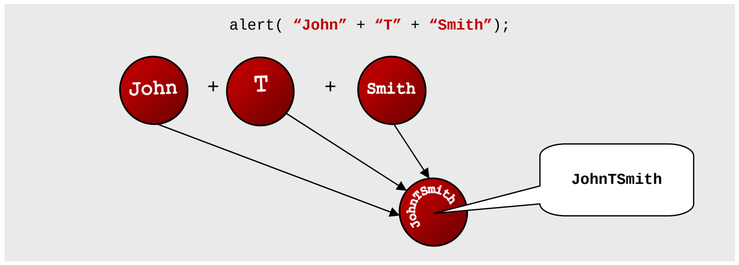
Adding two strings will result in the second string being added onto the end of the first string.

```
alert( "John" + "T" + "Smith");
```

You can not subtract a string.







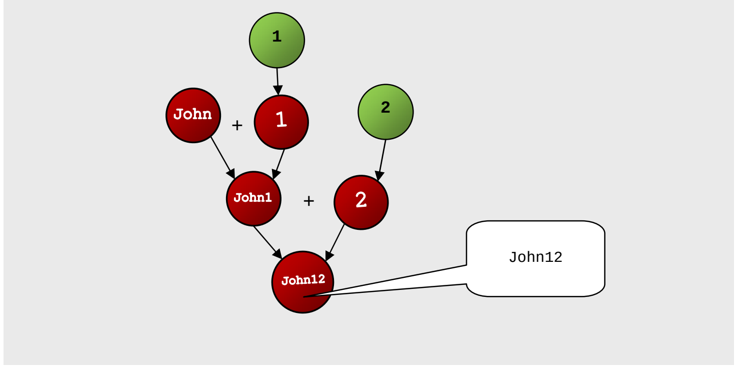
If you try to add a number to a string, the number will be treated as a string and simply added on to the end of the string.

```
alert( "John" + 1 + 2);
```

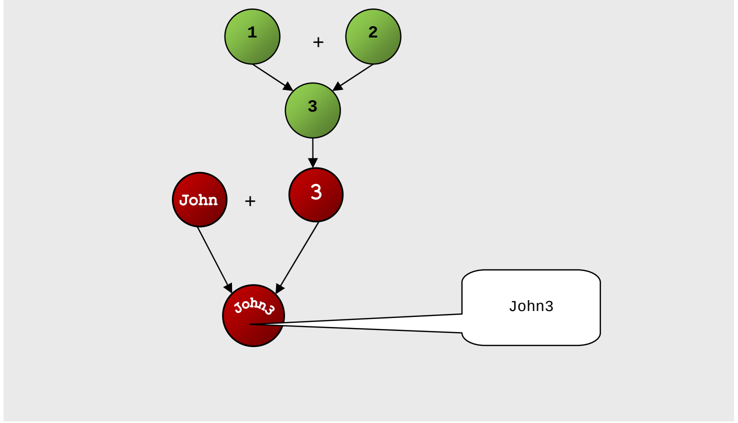
However, if you surround the number arithmetic with parentheses, the arithmetic will be performed first and then added to the end of the string.

```
alert( "John" + (1 + 2) );
```

```
alert("John" + 1 + 2);
```

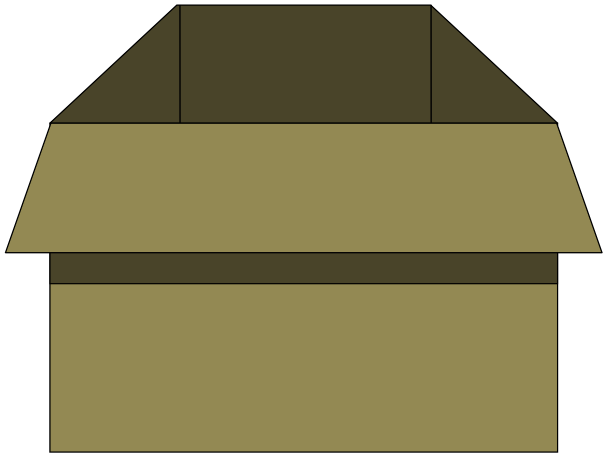


```
alert("John" + (1 + 2) );
```



Variables

Variables are like boxes that we can put objects into, so we'll use boxes to represent variables in the illustrations. In the code, variables are represented by small strings of text. Variable names must begin with a letter and can only contain letters, numbers and underscores (_).



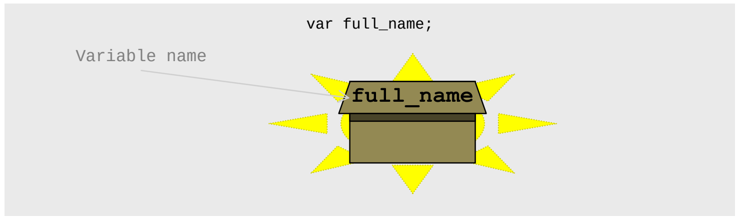
Before we can use variables, we have to create them. To do this in javaScript use the “var” keyword. Here we will create a variable named “`full_name`”.

```
var full_name;
alert(full_name);
```

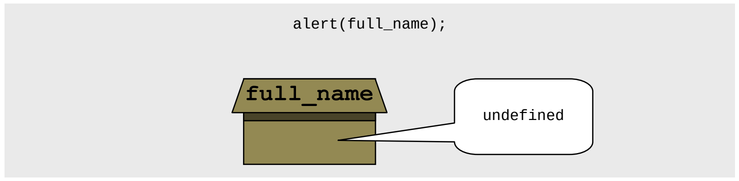
An empty variable that has nothing in it is “`undefined`”.

Variables do us no good if they are undefined, so we will need to put an object into the variable using the **“=“** operator. The object that is inside a variable is called the value.

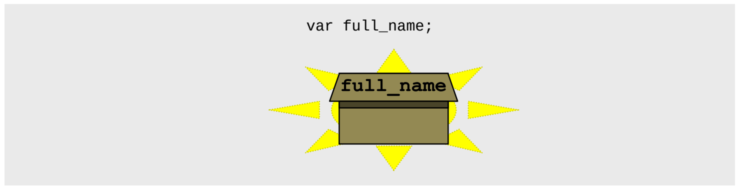
```
var full_name;
full_name = “John Smith”;
alert(full_name);
```



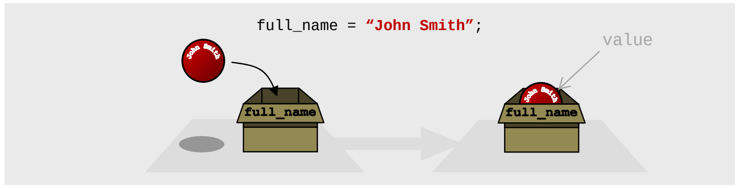
```
var full_name;
```



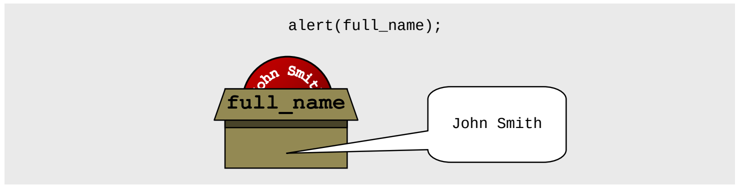
```
alert(full_name);
```



```
var full_name;
```



```
full_name = “John Smith”;
```



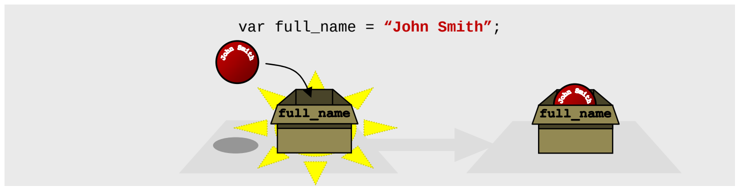
```
alert(full_name);
```

We can set a variable’s value as soon as it is created.

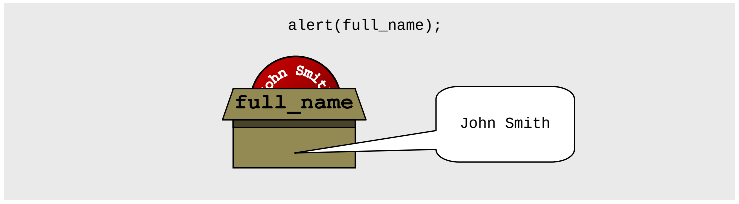
```
var full_name = “John Smith”;
alert(full_name);
```

Variables can hold numbers as well as strings.

```
var some_number = 12
alert(some_number);
```

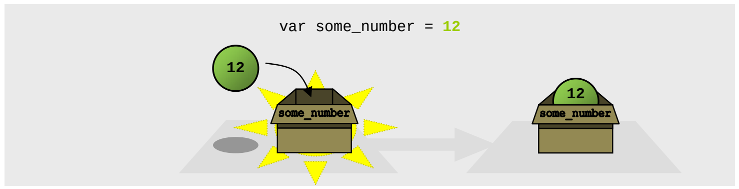


```
var full_name = “John Smith”;
```

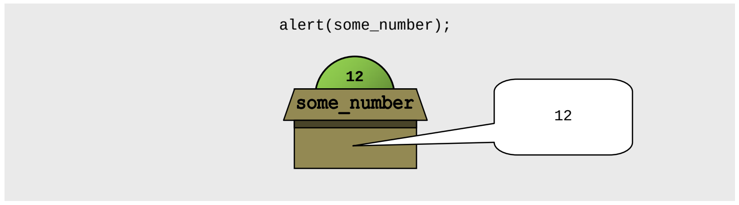


```
alert(full_name);
```

```
John Smith
```



```
var some_number = 12
```



```
alert(some_number);
```

```
12
```

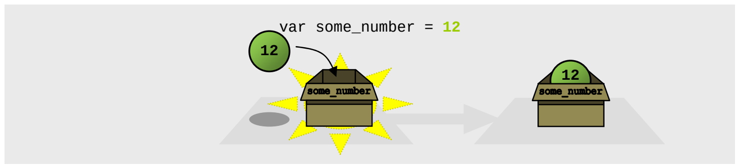
Once an object is inside a variable, we can do the same things to the variable that we can do to the object.

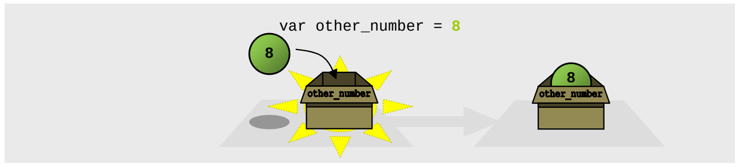
```
var some_number = 12
var other_number = 8
alert(some_number + other_number);
```

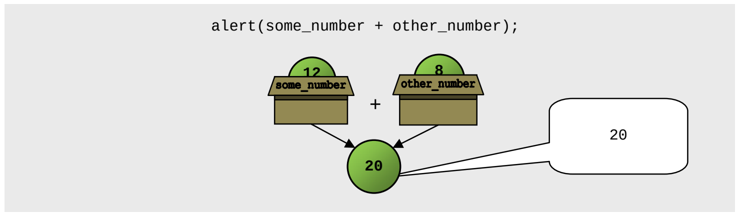
Because we can treat a variable as an object, we can even mix variables and non-variables.

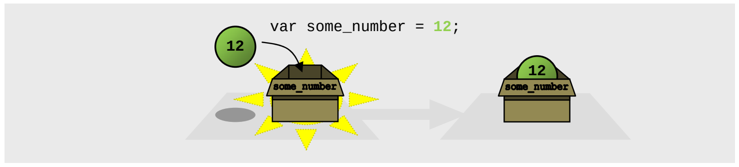
```
var some_number = 12;
alert("value of some_number: " +
   some_number );
```

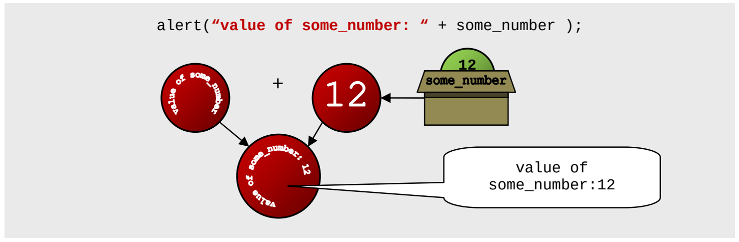
Remember: when adding a string to a number, the number is always changed into a string first.







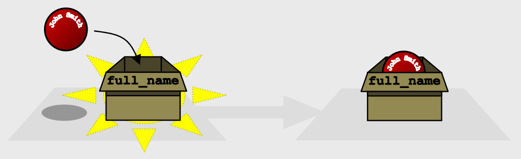




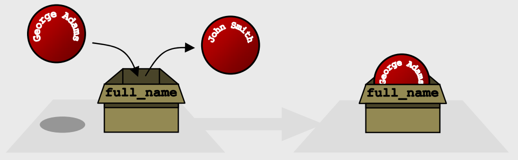
We can change the value of a variable by overwriting it. The original object will be replaced by the new object and the original object will be destroyed.

```
var full_name = “John Smith”;
full_name = “George Adams”;
alert(“full_name: “ + full_name);
```

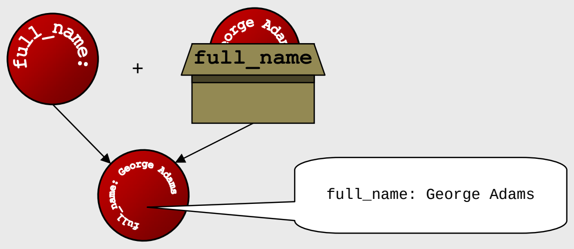
```
var full_name = “John Smith”;
```



```
full_name = “George Adams”;
```



```
alert(“full_name: “ + full_name);
```

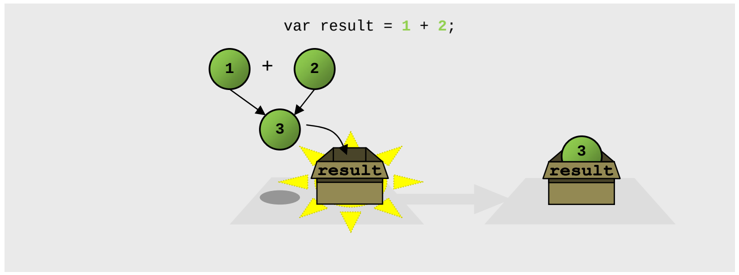


```
full_name: George Adams
```

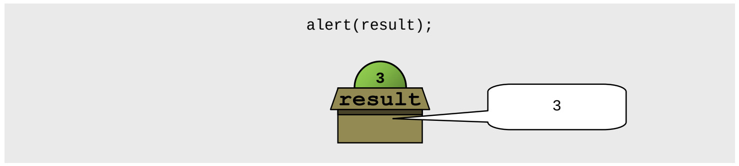
By putting an operation on the right side of the "=", we can put the result of that operation into the variable.

```
var result = 1 + 2;
alert(result);
result = "John" + "Smith";
alert(result);
```

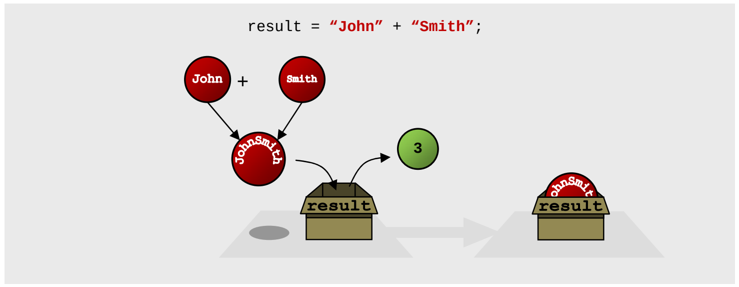
Remember: when overwriting a variable's value, the original value is destroyed.



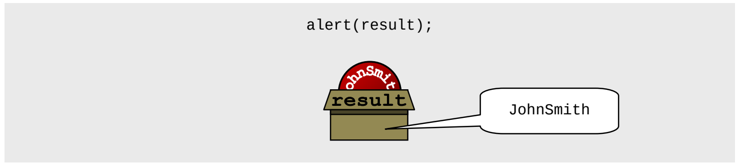
```
var result = 1 + 2;
```



```
alert(result);
```



```
result = "John" + "Smith";
```

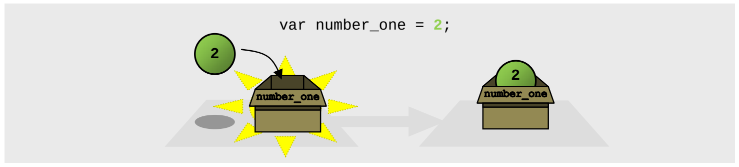


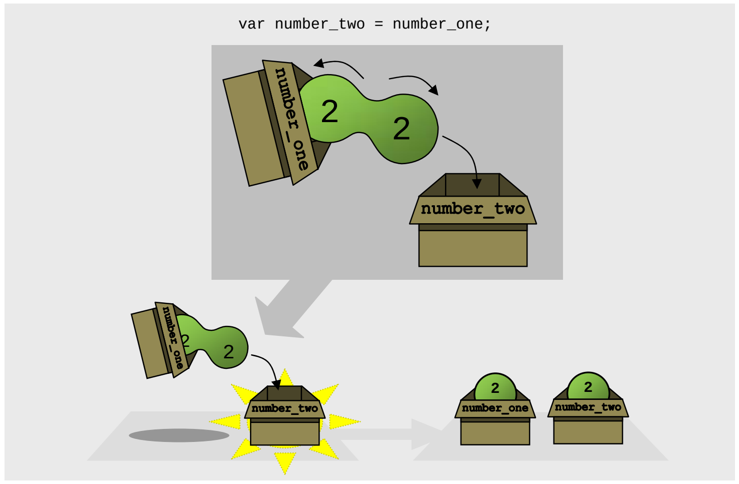
```
alert(result);
```

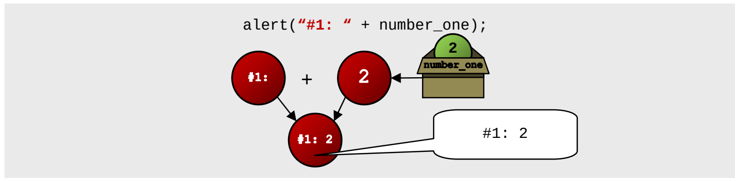
If we set the value of one variable to the value of another, it is COPIED to the new variable, and both contain the same value.

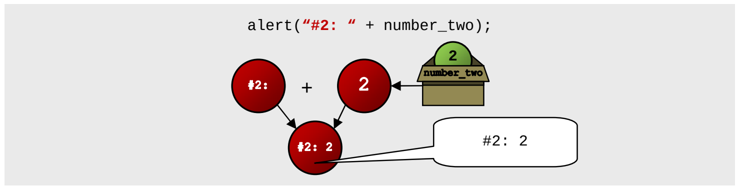
```
var number_one = 2;
var number_two = number_one;
alert("#1: " + number_one);
alert("#2: " + number_two);
```

Remember: when adding a string to a number, the number is always changed into a string first.



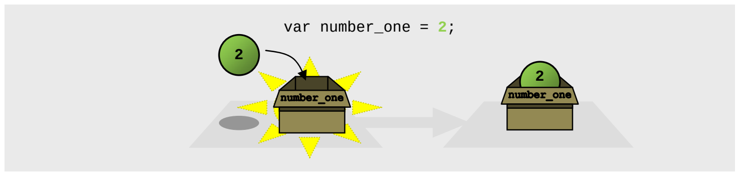




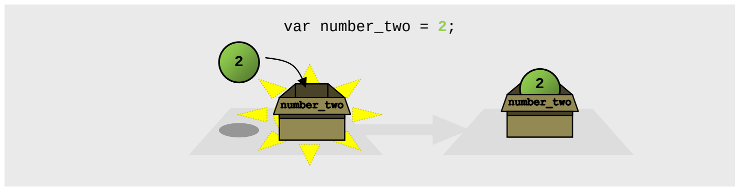


Because we can treat a variable exactly like the object that is inside it, we can perform operations on the variables and put the result of that operation into another variable.

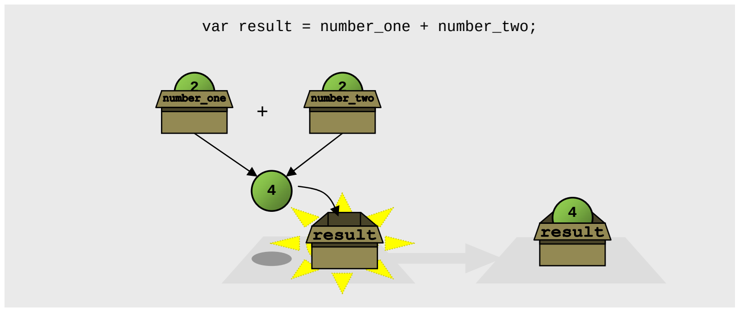
```
var number_one = 2;
var number_two = 2;
var result = number_one + number_two;
alert("Result: " + result);
```



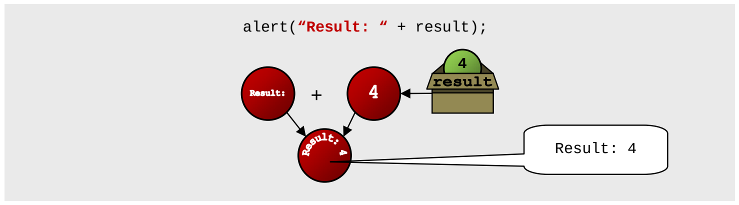
```
var number_one = 2;
```



```
var number_two = 2;
```



```
var result = number_one + number_two;
```



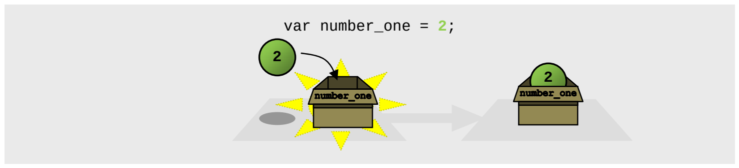
```
alert("Result: " + result);
```

```
Result: 4
```

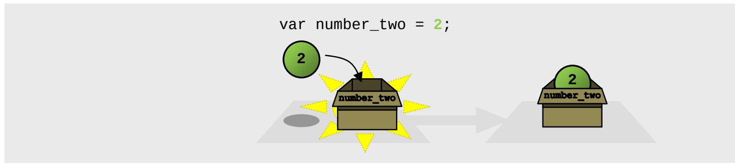
Everything on the right side of the "=" will happen BEFORE the result of the operation is put into the variable. Because of this, we can reuse variables .

```
var number_one = 2;
var number_two = 2;
number_one = number_one + number_two;
number_one = "#1: " + number_one;
alert( number_one );
```

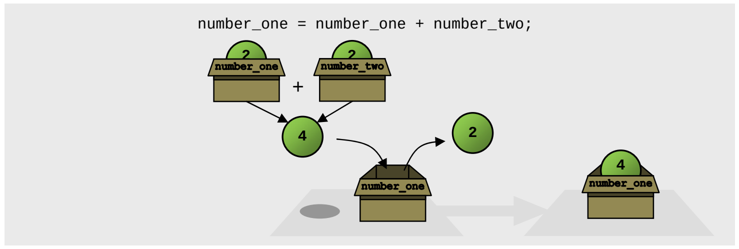
Remember: when overwriting a variable's value, the original value is destroyed.



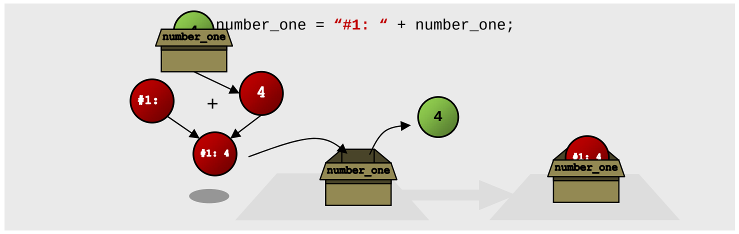
```
var number_one = 2;
```



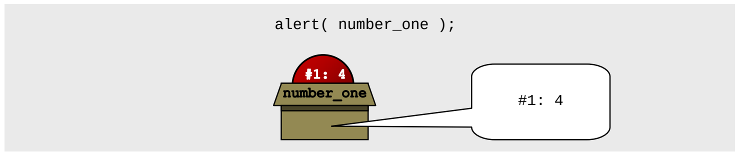
```
var number_two = 2;
```



```
number_one = number_one + number_two;
```



```
number_one = "#1: " + number_one;
```

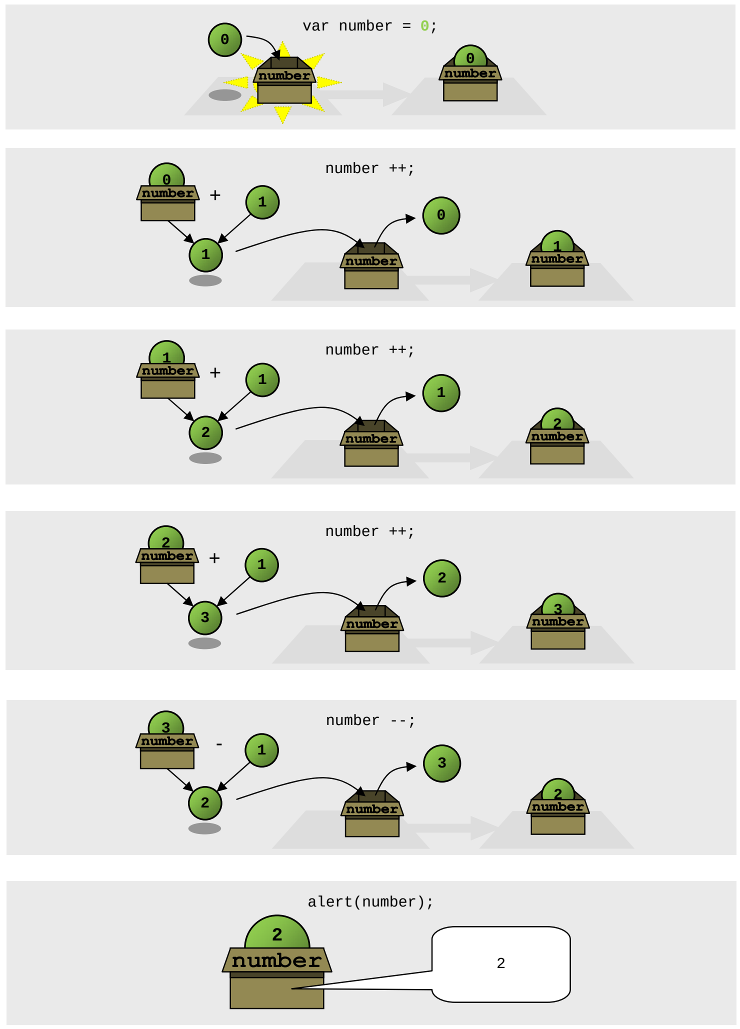


```
alert( number_one );
```

```
#1: 4
```

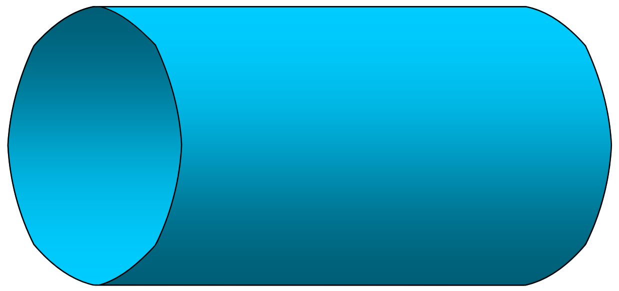
Lastly, there are special operators that we can use on a number variable. “**++**” and “**--**" which add 1 to the value and subtract 1 from the value, respectively.

```
var number = 0;
number ++;
number ++;
number ++;
number --;
alert(number);
```



Functions

If variables are the "nouns" of programming, then functions would be the “verbs". Simply put, a function is something that does something. You put some objects into the function and the function will do something with those objects. In the code, a function is expressed by a word followed by a pair of parentheses which contain a list of objects. In our illustrations, we'll use a tube shape to represent functions.



You've been using a function this whole time to create the popups which show the output of your code.

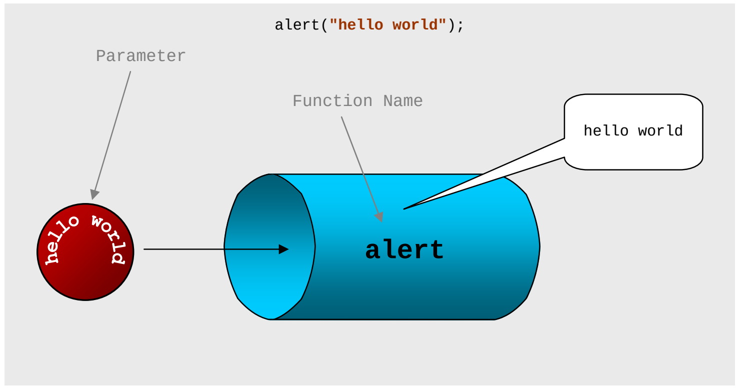
```
alert("hello world");
```

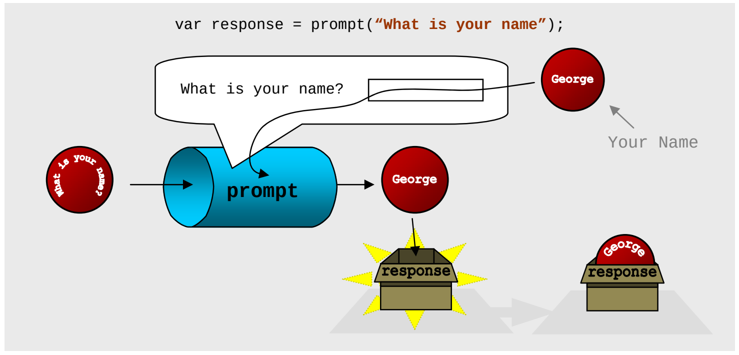
The word "alert" is the function name. The string between the parentheses (in this case "hello world") is called the 'parameter'. The **alert** function will change the parameter into a string, regardless of its type, and create a pop up based on that string.

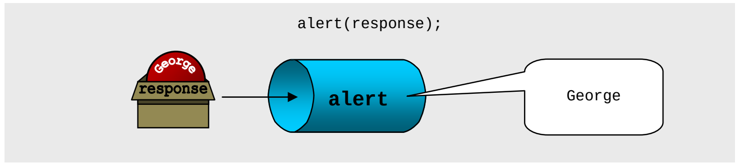
In addition to simply 'doing something', many functions return objects. When you put parameters into a function, the function will usually do something to those parameters and then spit out a new object that can be put into a variable. Lets take a look at another basic function: **prompt()**. This function takes in a string as the prompt and returns the value that is entered so that it can be put into a variable.

For these examples, lets just pretend your name is **George**.

```
var response = prompt("What is your
   name");
alert(response);
```







Some functions can accept multiple parameters. Prompt can accept an optional second parameter for the default value to be put in the textbox of the prompt.

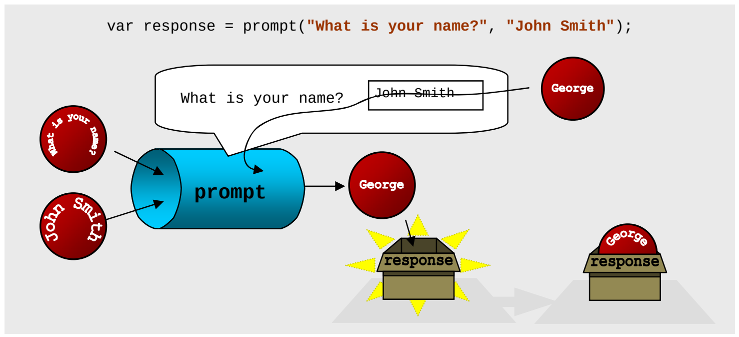
```
var response = prompt("What is your
   name?", "John Smith");
alert(response);
```

Note that this time, the input box is already filled with the second parameter, `"John Smith"`, before you enter your name.

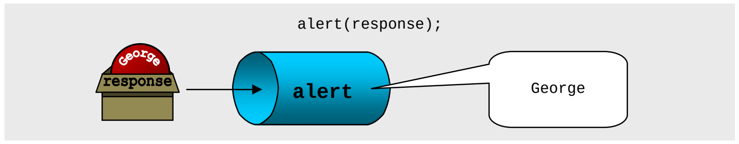
Lets run the same code once more, but this time, click the **Cancel** button instead of **OK**.

Because you clicked **Cancel**, anything you entered into the box was disregarded and a **null** value is returned. A **null** value is basically a blank object. A variable with **null** in it is not undefined because it has *something* in, but that something has no value. It will be treated as **0** if used as a number, but will be treated as **"null"** when used as a string.

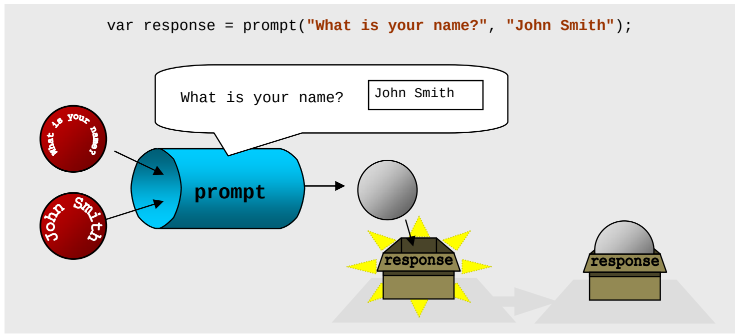
In this case, **null** gets passed directly into **alert**, becomes a string and a popup is created.



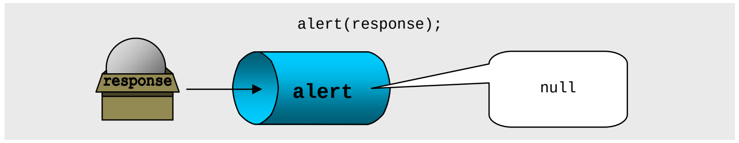
```
var response = prompt("What is your name?", "John Smith");
```



```
alert(response);
```



```
var response = prompt("What is your name?", "John Smith");
```



```
alert(response);
```

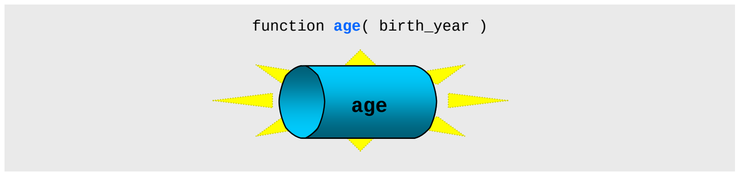
Custom functions make it possible to group large chunks of complex code together in a way which allows you to reuse it later on, greatly reducing the amount of code you need to write.

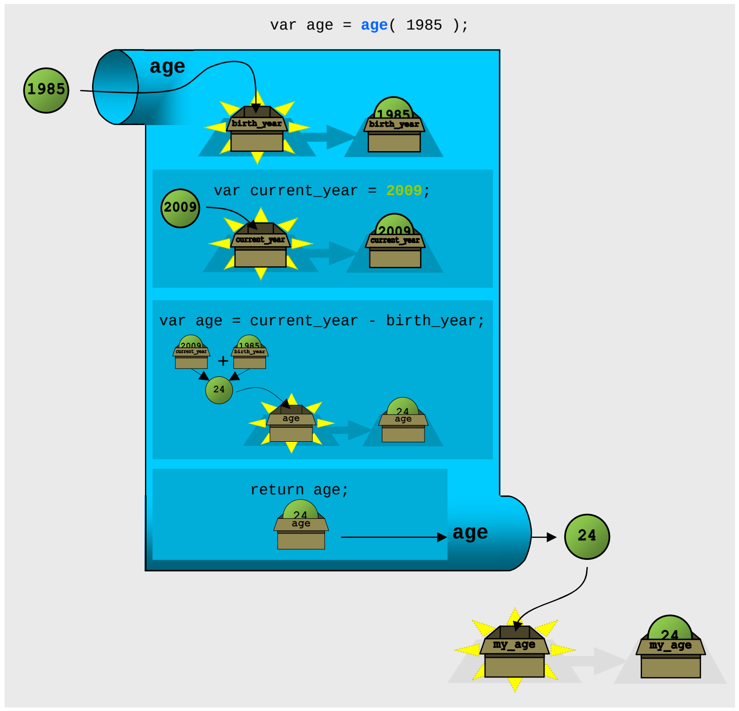
To define the function, we use the **function** keyword, followed by the function name, and then a list of parameters names. Each name in the parameter list will become a variable that can only be used inside the function. When we pass parameters into a function, they are immediately put into the variables in the order that they've been defined.

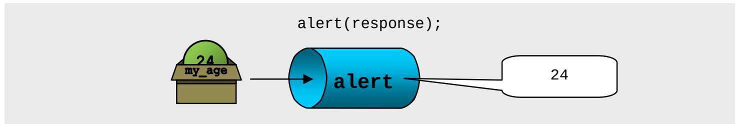
The code in between the **curly brackets** is the code that makes up the function and will not run until you call the function.

At the end of the function definition, we can use the **return** keyword to tell the function what exactly we are going to return. Only one object can be returned by a function.

```
function age( birth_year )
{
   var current_year = 2009;
   var age = current_year - birth_year;
   return age;
}

var my_age = age( 1985 );
alert( my_age );
```





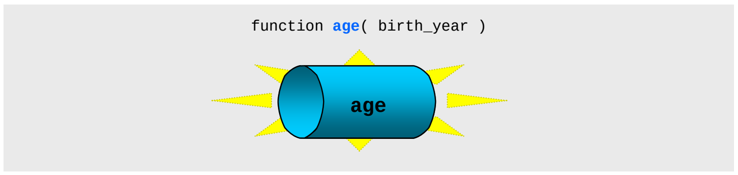


Now that we know how to use objects, variables, and functions, lets put together everything we know to write a simple script.

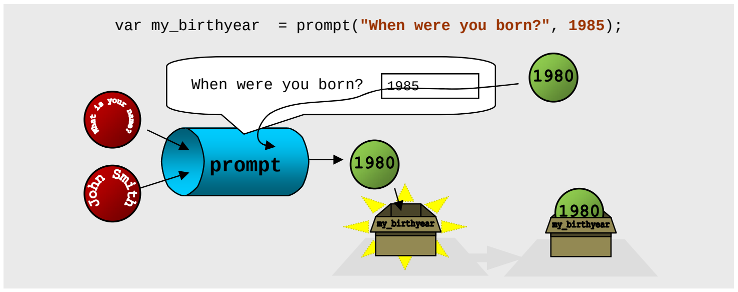
For this example, lets pretend you were born in **1980**.

```
function age( birth_year )
{
   var current_year = 2009;
   var age = current_year - birth_year;
   return age;
}

var my_birthyear = prompt( “When were
   you born?”, 1985 );

var my_age = age( my_birthyear );
alert( “You are “ + my_age + “ years
   old” );
```

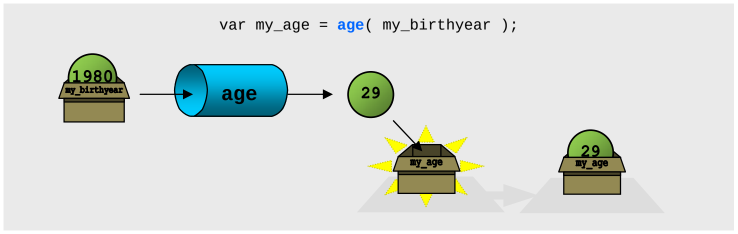
Congratulations.
You’ve just learned how to program.



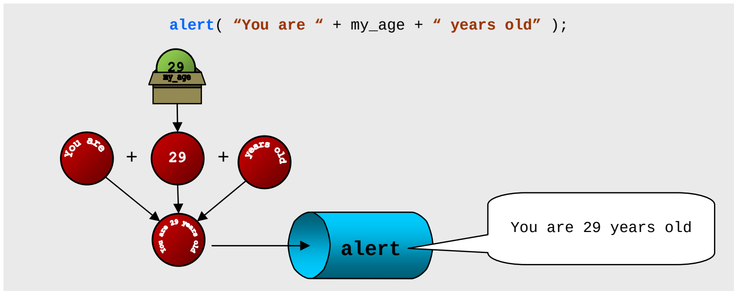
```
function age( birth_year )
```



```
var my_birthyear  = prompt("When were you born?", 1985);
```



```
var my_age = age( my_birthyear );
```



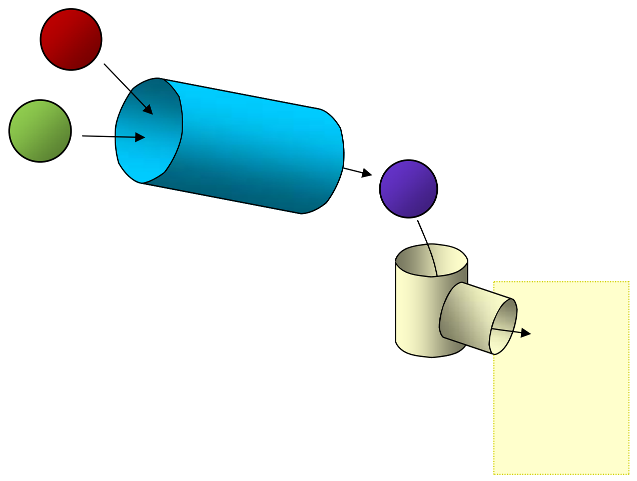
```
alert( “You are “ + my_age + “ years old” );
```

Booleans and Conditionals

Booleans and **Conditionals** can be used to control the ‘flow’ of a script. Previously, the scripts demonstrated start at the top and end at the bottom, as one would expect.

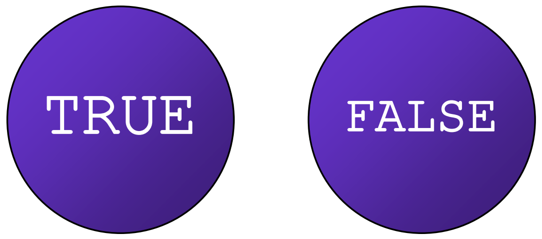
So far, the only things that we’ve looked at so far that can sort of control the flow of a script are functions. Functions are defined before they are used, but they always run every time they are called.

Sometimes you may want to run parts of code only if certain conditions are true. This is where Booleans and conditionals come in handy. They allow us to look at something and then run specific code depending on whether or not it is true.



Booleans

Booleans are special types that only represent True or False values. In code booleans are represented by "**true**" and "**false**" keywords. For our purposes, booleans will be expressed as purple spheres with the value of TRUE or FALSE written over them.

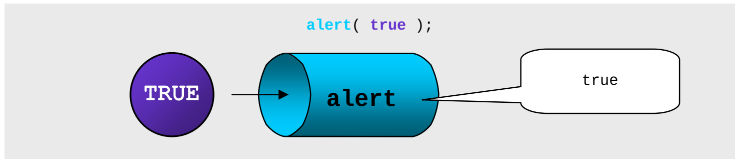


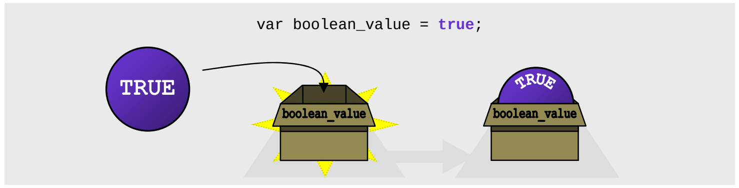
Booleans, like other types, can be used as constant, unchanging objects...

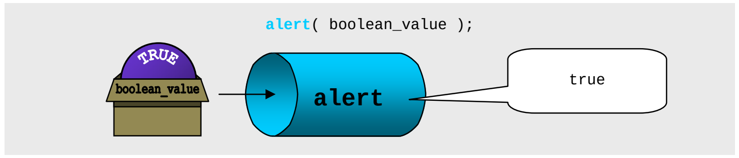
```
alert( true );
```

Or they can be put into variables.

```
var boolean_value = true;
alert( boolean_value );
```



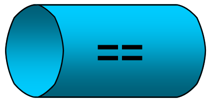




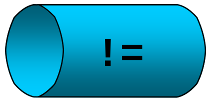
Comparisons

Booleans can be created by using the comparison operators. In our illustrations, comparisons will use the same 'function tube' that we used before, but they will have the operator written in place of the function name. These operators are like special functions that take parameters on each side, compare them and return whether or not the comparison is true. This returned value can be used just as you would the result of a regular function.

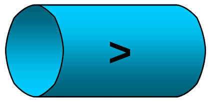
== (double equal signs)
Are two parameters **equal**?



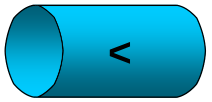
!= (exclamation point and equal sign)
Are two parameters **NOT equal**?



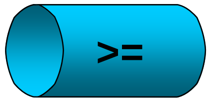
> (greater than sign)
Is the left side **greater than** the right side?



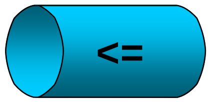
< (less than sign)
Is the left side **less than** the right side?



>= (greater than sign and equal sign)
Is the left side **greater than or equal** to the right side?



<= (less than sign and equal sign)
Is the left side **less than or equal** to the right side?



== (double equal signs)

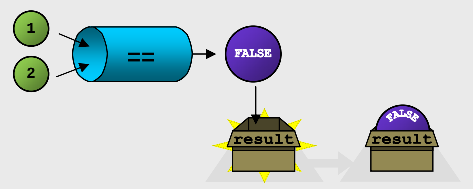
Are the two parameters equal?

```
var result = 1 == 2;
alert( “1 equals 2? “ + result );
result = 2 == 2;
alert( “2 equals 2? “ + result );
result = 3 == 2;
alert( “3 equals 2? “ + result );
```

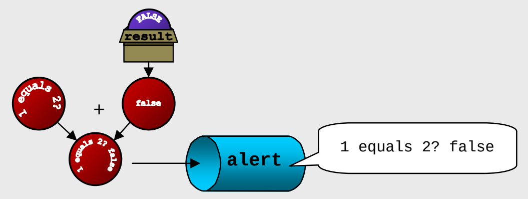
Remember: just like numbers and null values, when being added to a string, Booleans will be turned into strings before the addition.

Remember: when overwriting a variable’s value, the original value is destroyed.

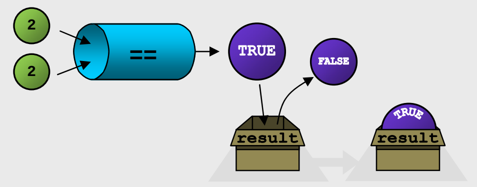
```
var result = 1 == 2;
```



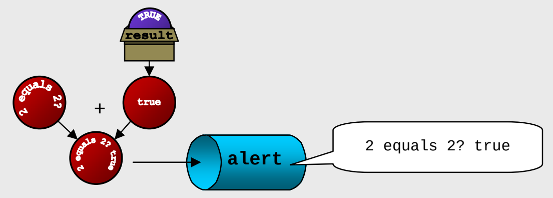
```
alert( “1 equals 2? “ + result );
```



```
result = 2 == 2;
```



```
alert( “2 equals 2? “ + result );
```

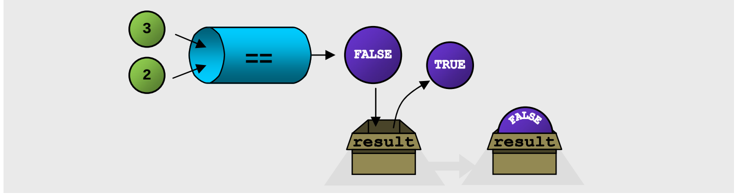


!= (exclamation point and equal sign)

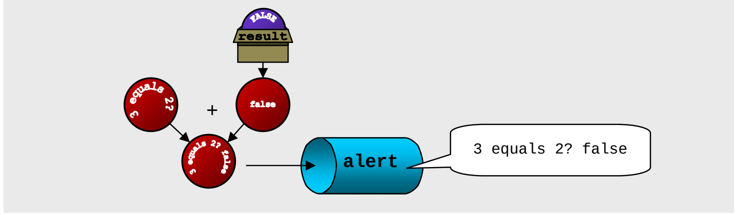
Are the two parameters NOT equal?

```
var result = 1 != 2;
alert( “1 does not equal 2? “ +
  result );
result = 2 != 2;
alert( “2 does not equal 2? “ +
  result );
result = 3 != 2;
alert( “3 does not equal 2? “ +
  result );
```

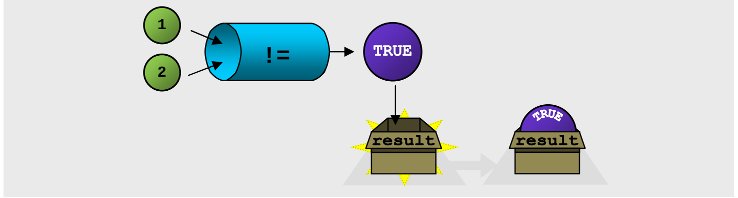
```
result = 3 == 2;
```



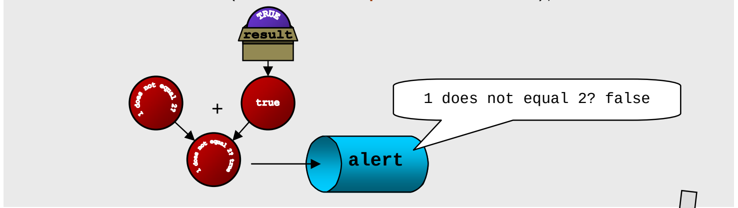
```
alert( “3 equals 2? “ + result );
```

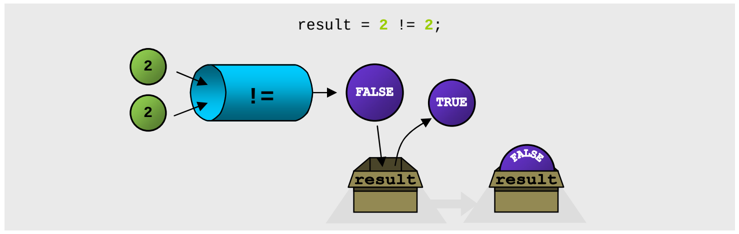


```
var result = 1 != 2;
```

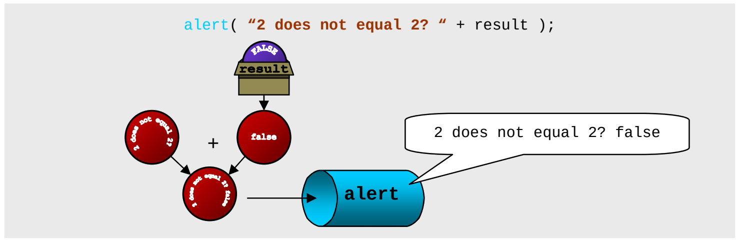


```
alert( “1 does not equal 2? “ + result );
```



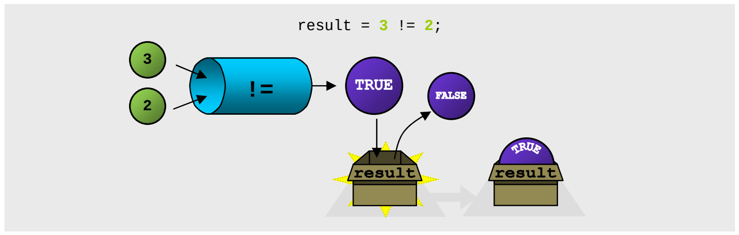


```
result = 2 != 2;
```

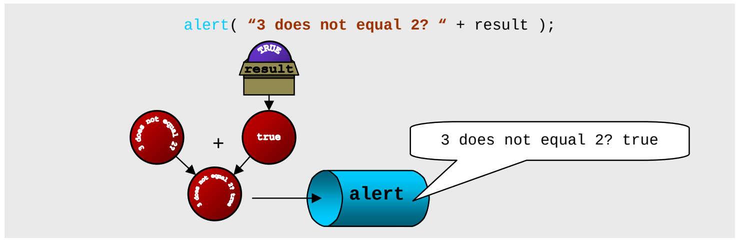


```
alert( "2 does not equal 2? " + result );
```

```
2 does not equal 2? false
```



```
result = 3 != 2;
```



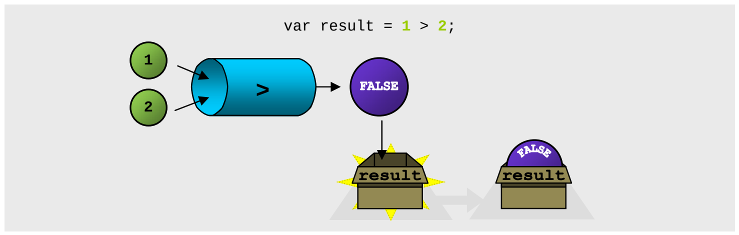
```
alert( "3 does not equal 2? " + result );
```

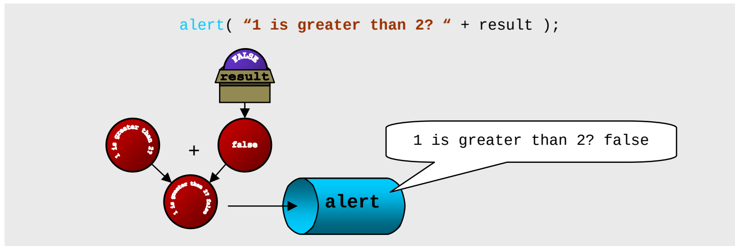
```
3 does not equal 2? true
```

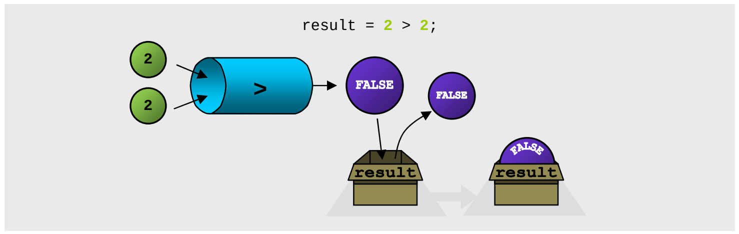
> (greater than sign)

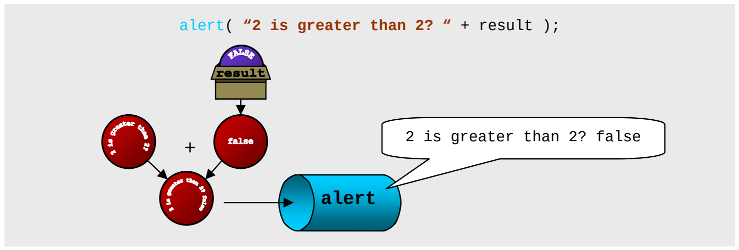
Is the left parameter greater than the right?

```
var result = 1 > 2;
alert( “1 is greater than 2? “ +
  result );
result = 2 > 2;
alert( “2 is greater than 2? “ +
  result );
result = 3 > 2;
alert( “3 is greater than 2? “ +
  result );
```




alert(“1 is greater than 2? “ + result);
FALSE
result
+
false
alert
1 is greater than 2? false



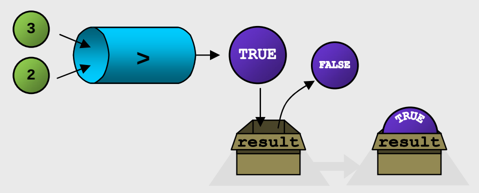

alert(“2 is greater than 2? “ + result);
FALSE
result
+
false
alert
2 is greater than 2? false

< (less than sign)

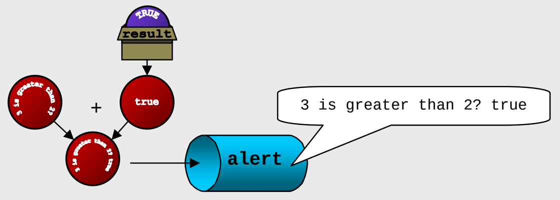
Is the left parameter less than the right?

```
var result = 1 < 2;
alert( "1 is less than 2? " + result );
result = 2 < 2;
alert( "2 is less than 2? " + result );
result = 3 < 2;
alert( "3 is less than 2? " + result );
```

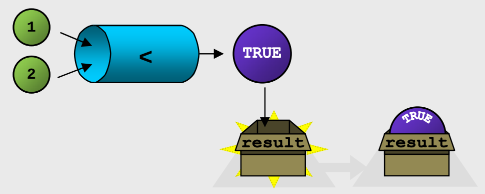
```
result = 3 > 2;
```



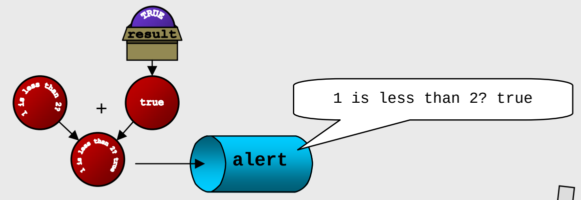
```
alert( "3 is greater than 2? " + result );
```

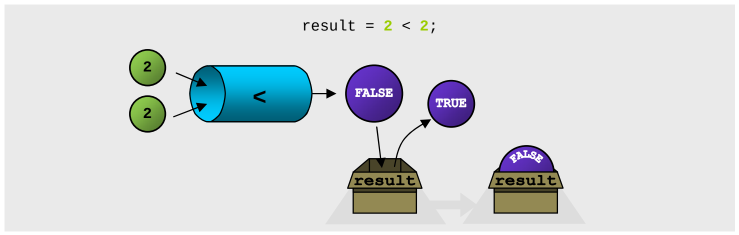


```
var result = 1 < 2;
```

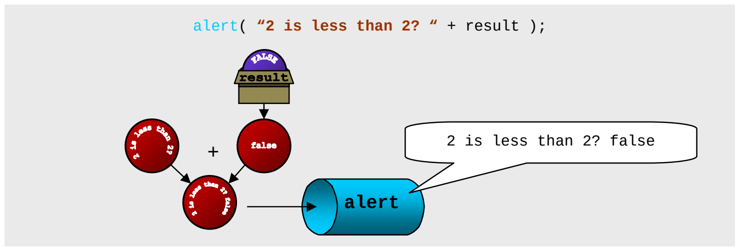


```
alert( "1 is less than 2? " + result );
```



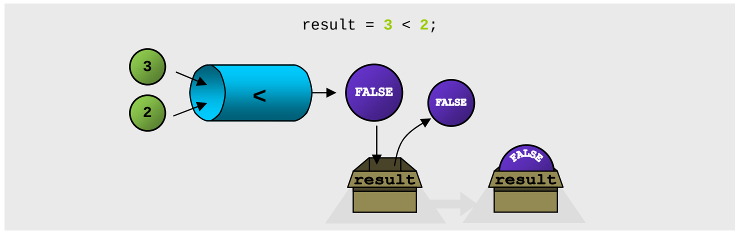


```
result = 2 < 2;
```

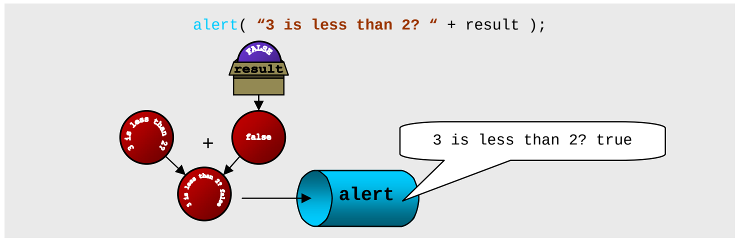


```
alert( “2 is less than 2? “ + result );
```

```
2 is less than 2? false
```



```
result = 3 < 2;
```



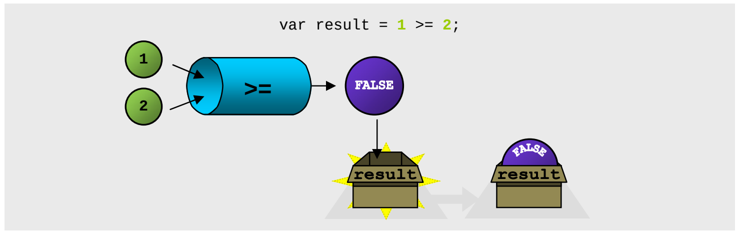
```
alert( “3 is less than 2? “ + result );
```

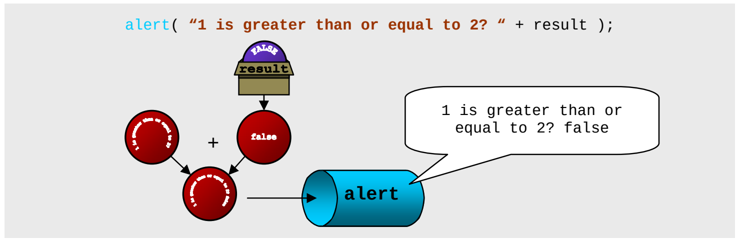
```
3 is less than 2? true
```

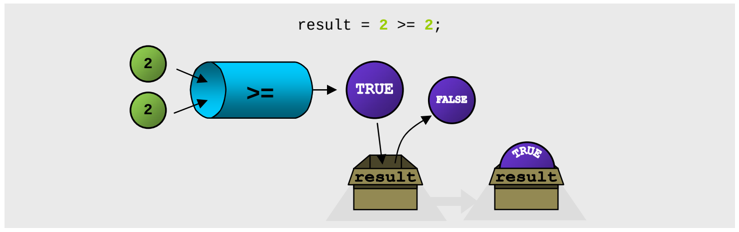
>= (greater than sign and equal sign)

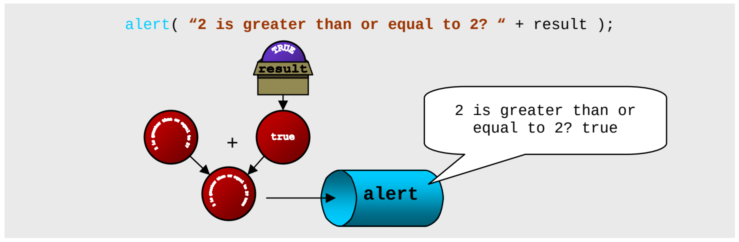
Is the left parameter greater than or equal to the right?

```
var result = 1 >= 2;
alert( “1 is greater than or equal to
   2?“ + result );
result = 2 >= 2;
alert( “2 is greater than or equal to
   2?“ + result );
result = 3 >= 2;
alert( “3 is greater than or equal to
   2?“ + result );
```




alert(“1 is greater than or equal to 2? “ + result);
FALSE
result
+
false
1 is greater than or equal to 2? false
alert

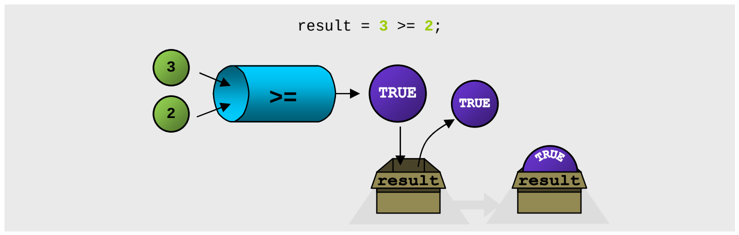



alert(“2 is greater than or equal to 2? “ + result);
TRUE
result
+
true
2 is greater than or equal to 2? true
alert

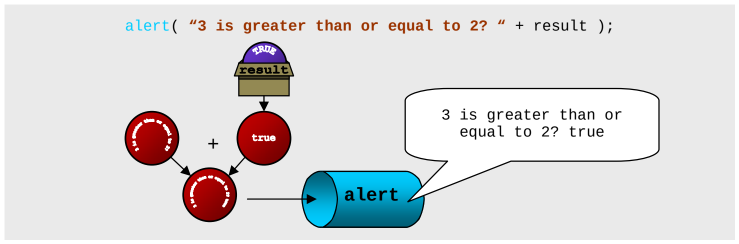
<= (less than sign and equal sign)

Is the left parameter less than or equal to the right?

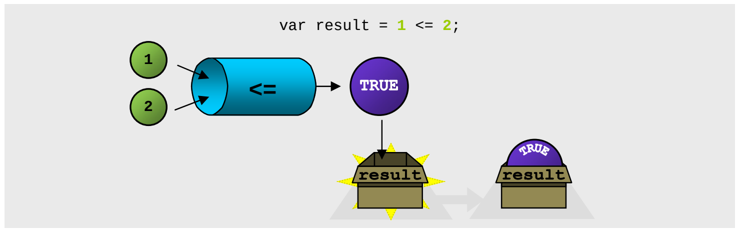
```
var result = 1 <= 2;
alert( “1 is less than or equal to 2? “
   + result );
result = 2 <= 2;
alert( “2 is less than or equal to 2? “
   + result );
result = 3 <= 2;
alert( “3 is less than or equal to 2? “
   + result );
```



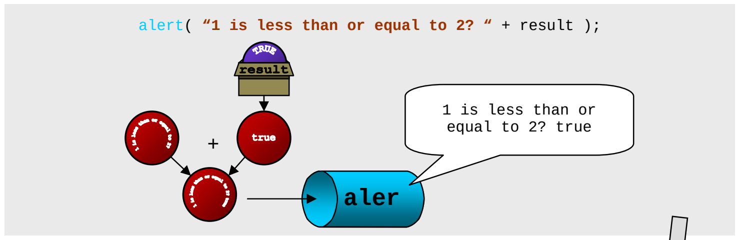
```
result = 3 >= 2;
```



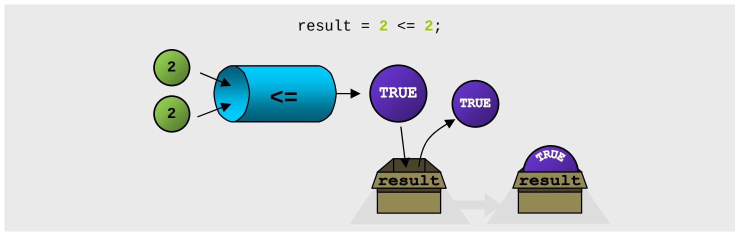
```
alert( “3 is greater than or equal to 2? “ + result );
```



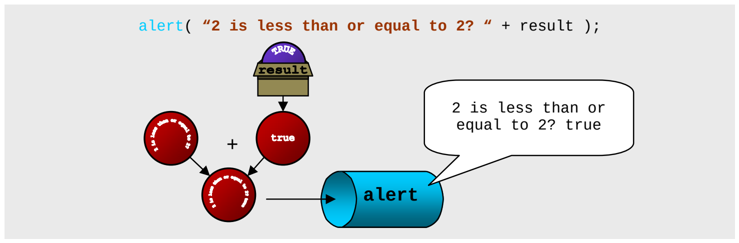
```
var result = 1 <= 2;
```



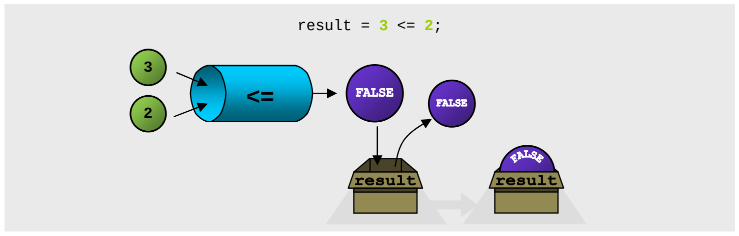
```
alert( “1 is less than or equal to 2? “ + result );
```



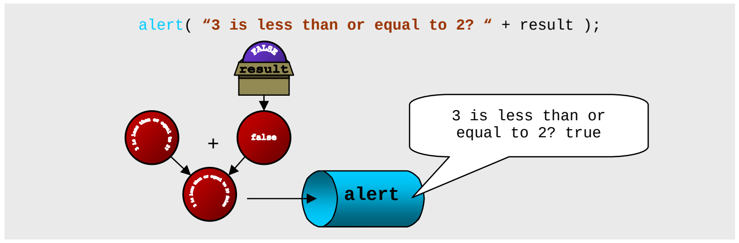
```
result = 2 <= 2;
```



```
alert( "2 is less than or equal to 2? " + result );
```



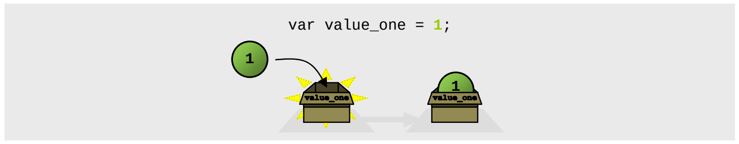
```
result = 3 <= 2;
```



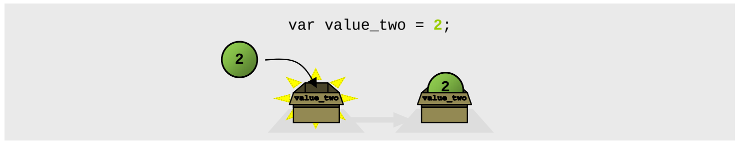
```
alert( "3 is less than or equal to 2? " + result );
```

And of course, as with everything else, we can use variables with the comparison operators.

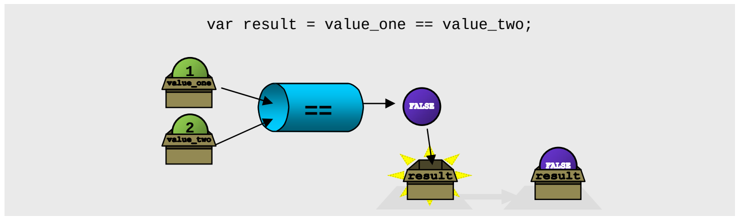
```
var value_one = 1;
var value_two = 2;
var result = value_one == value_two;
alert("Does value_one equal value_two? "
   + result);
result = value_one <= value_two;
alert("Is value_one less than or equal
   to value_two? " + result);
```



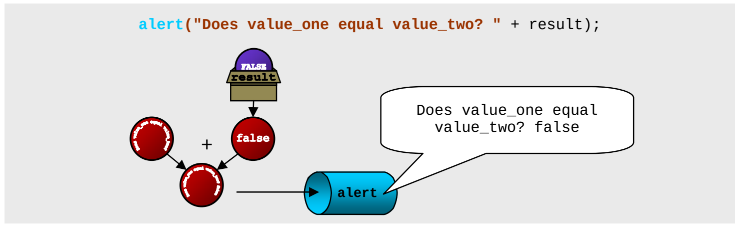
```
var value_one = 1;
```



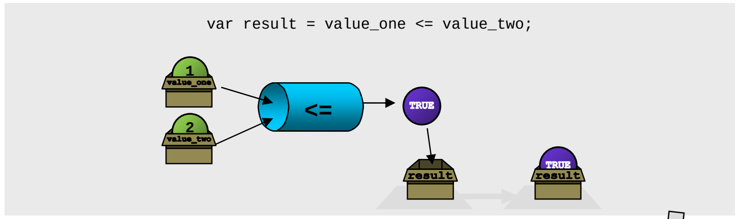
```
var value_two = 2;
```



```
var result = value_one == value_two;
```



```
alert("Does value_one equal value_two? " + result);
```

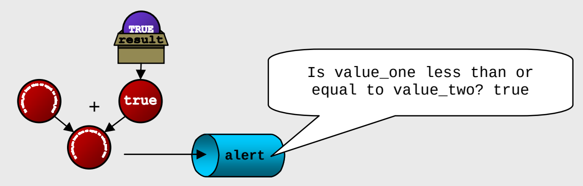


```
var result = value_one <= value_two;
```

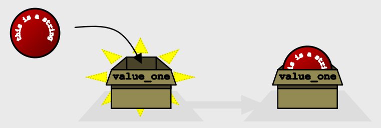
Thus far, we've only talked about numbers, but comparisons can occur between pretty much any kind of object;

```
var value_one = "this is a string";
alert( value_one == "this is a
   string" );
alert( value_one == 2 );
```

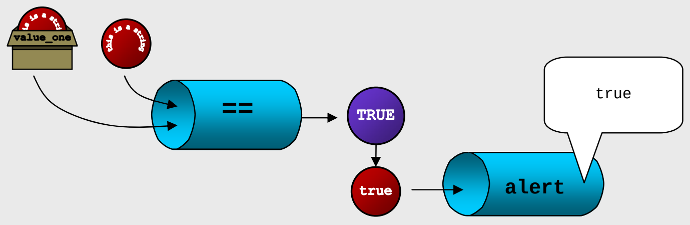
```
alert("Is value_one less than or equal to value_two? " + result);
```



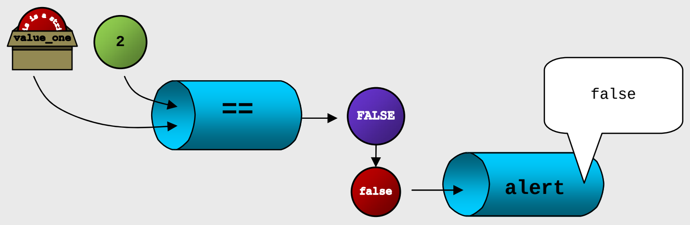
```
var value_one = "this is a string";
```



```
alert( value_one == "this is a string" );
```



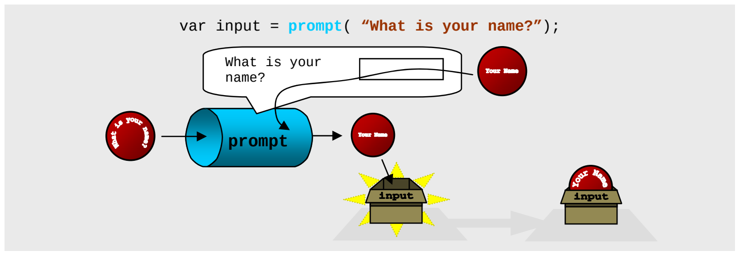
```
alert( value_one == 2 );
```



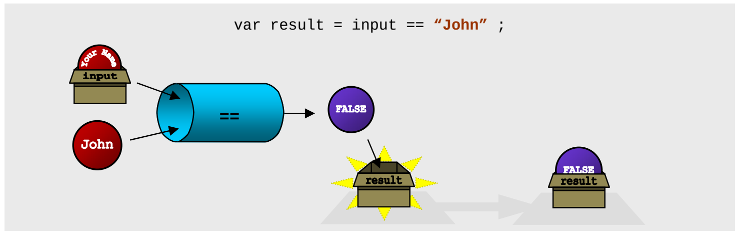
Booleans and comparisons are useful for checking user input. Try running the following code several times. The first, time try entering your own name.

```
var input = prompt( “What is your
   name?”);
var result = input == “John” ;
alert( input + “ is John?” + result );
```

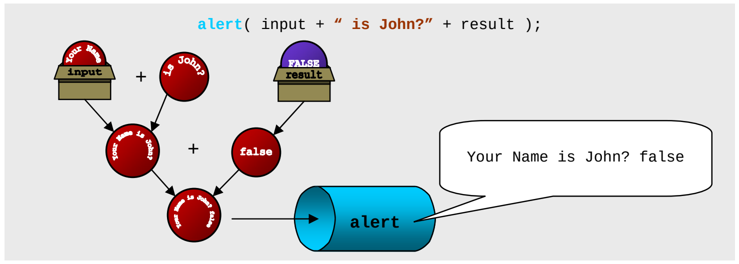
Now, try running it again but enter **“John”** instead of your own name.



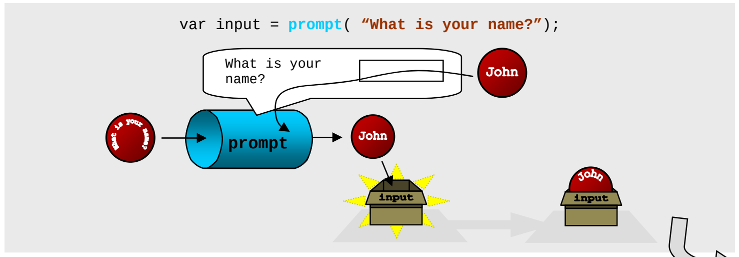
```
var input = prompt( “What is your name?”);
```



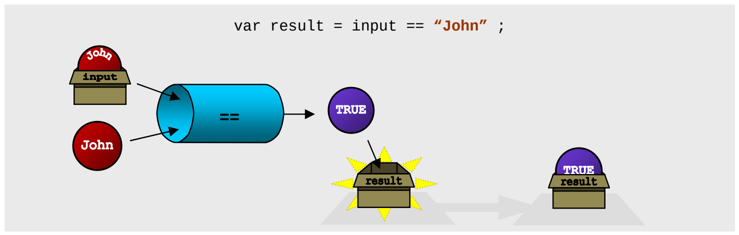
```
var result = input == “John” ;
```



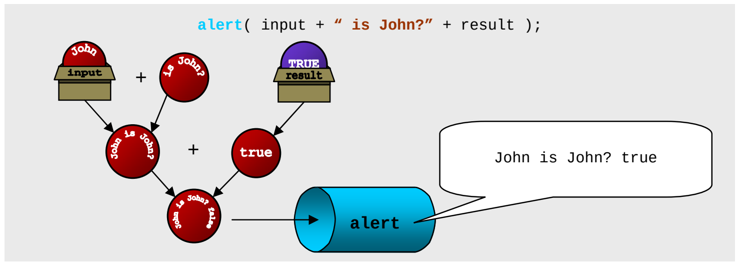
```
alert( input + “ is John?” + result );
```



```
var input = prompt( “What is your name?”);
```



```
var result = input == "John" ;
```

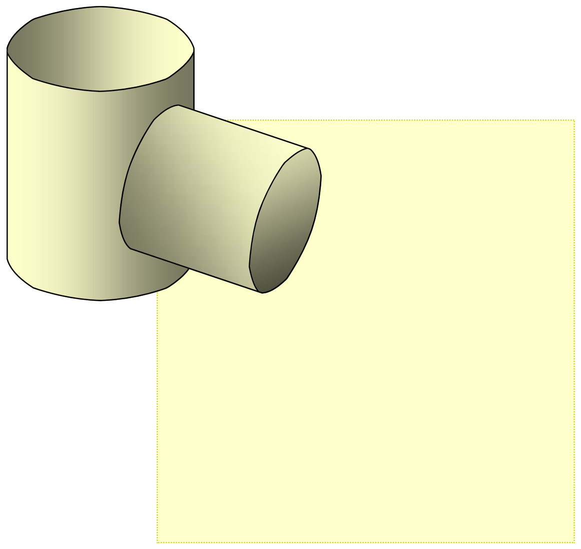


```
alert( input + " is John?" + result );
```

```
John is John? true
```

Conditionals

Conditionals allow you to run a bit of code only if a certain requirement or 'condition' is true. In the code, a conditional is created by using the **"if"** keyword followed by a set of parentheses containing a condition and a set of curly brackets { } containing the code that should be run if the condition is true. In our illustrations, we'll use a light yellow "T" pipe shape that will lead into a light yellow block if the condition is true.

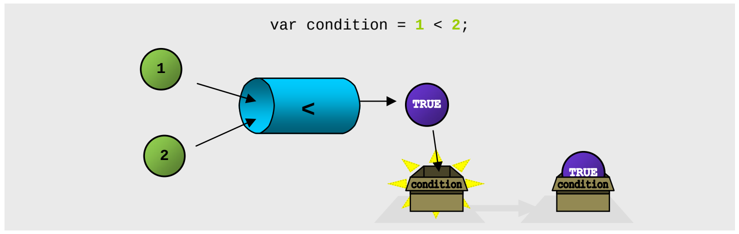


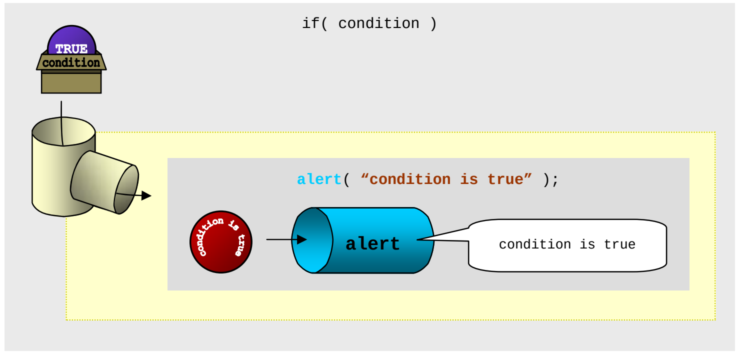
Lets take a look at a few simple conditions.

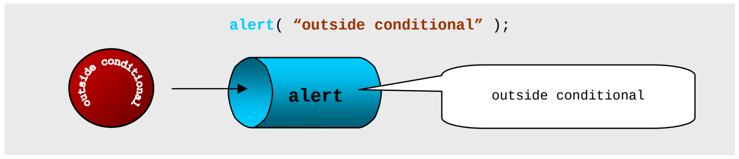
In this example, the condition is true, which causes the code inside the conditional block to run.

Note: you can not have a semicolon at the end of the conditional statement. If you add a semi colon, the following code block will not be recognized as part of the conditional.

```
var condition = 1 < 2;
if( condition )
{
   alert( “condition is true” );
}
alert( “outside conditional” );
```



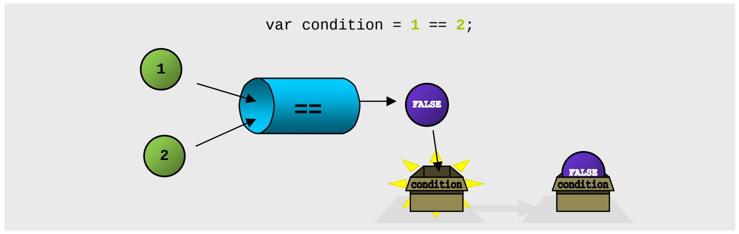


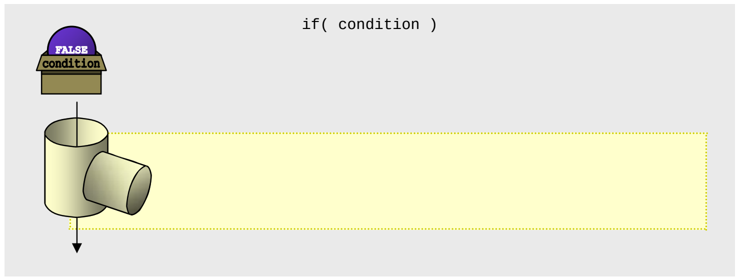


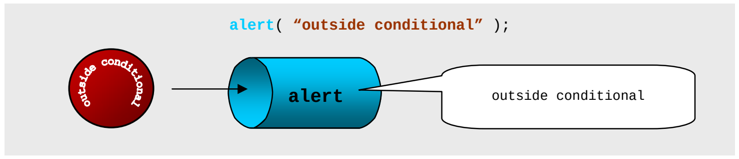
This next example is essentially the same. The only difference is that this time, the condition is false. This time, the code inside the conditional block will be completely ignored.

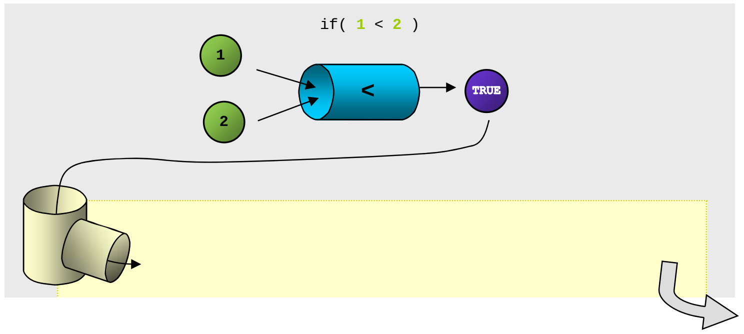
```
var condition = 1 == 2;
if( condition )
{
   alert( "condition is true" );
}
alert( "outside conditional" );
```

To make coding easier, most comparisons take place in the **if** statement directly. If done this way, the statement can be read aloud as "**if one is less than two, do what is in the conditional block**". This makes the code much easier to comprehend.





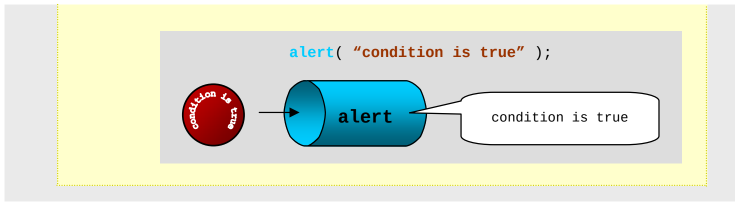


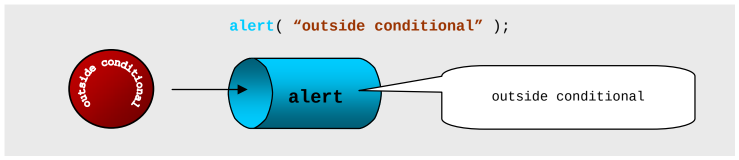


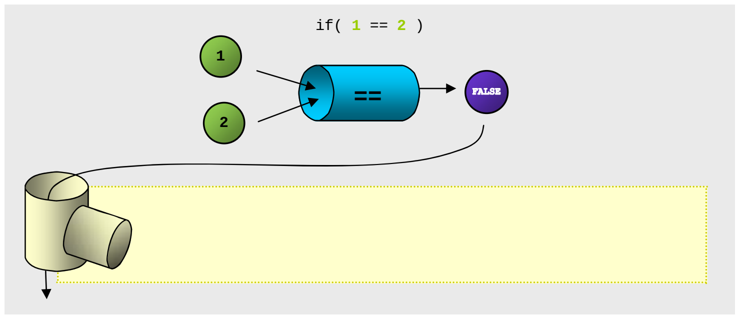
```
if( 1 < 2 )
{
   alert( “condition is true” );
}
alert( “outside conditional” );
```

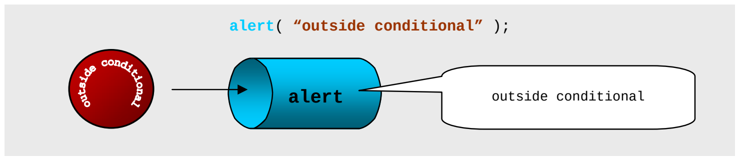
This example is the same as the last, but with a comparison that evaluates to **false**. It can be read as “**if one is equal to two, do what is in the conditional block**”. This statement does not make sense which is why it by-passes the code inside the conditional block.

```
if( 1 == 2 )
{
   alert( “condition is true” );
}
alert( “outside conditional” );
```





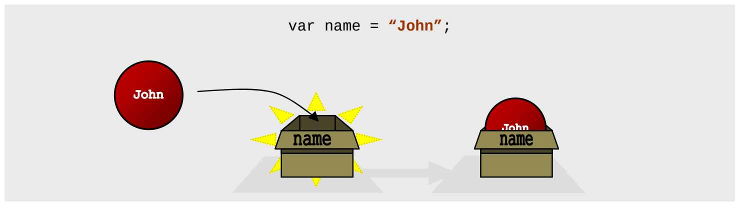




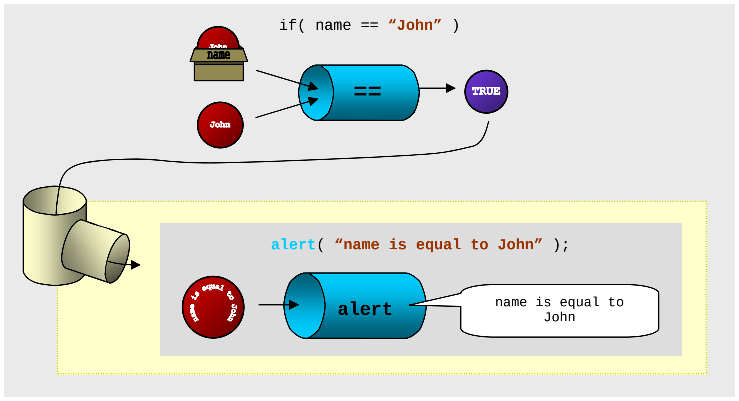
And again, of course, we can still use variables and strings instead of numbers.

```
var name = “John”;
if( name == “John” )
{
   alert( “name is equal to John” );
}
if( name != “John” )
{
   alert( “name is NOT John” );
}
alert( “outside conditional” );
```

Remember: the comparison != tries to determine if the two parameters are NOT EQUAL to each other.

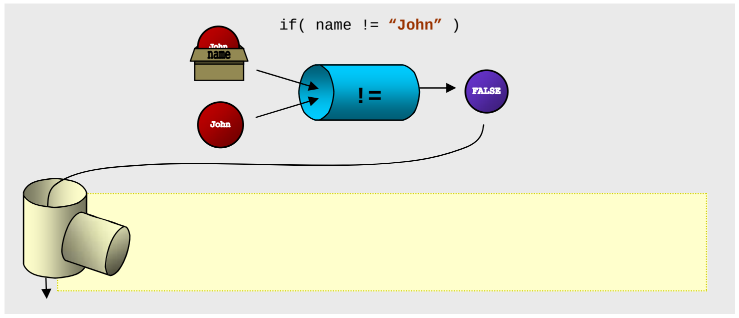


```
var name = “John”;
```



```
if( name == “John” )
```

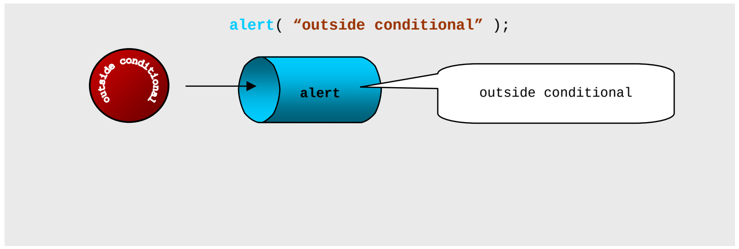
```
alert( “name is equal to John” );
```

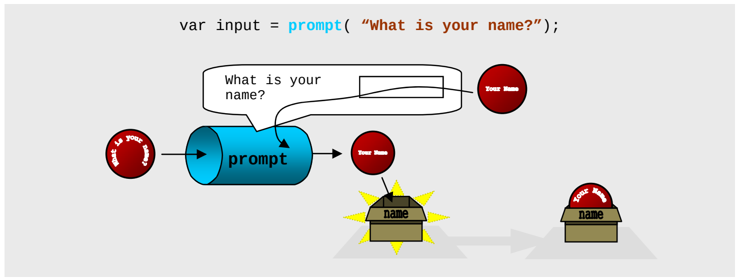


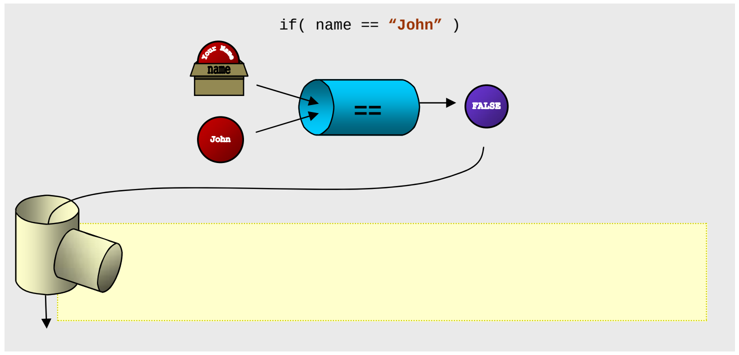
```
if( name != “John” )
```

We can now spice this script up by gathering some user input with prompt and use that input in the conditional.

```
var name = prompt( “What is your
   name?” );
if( name == “John” )
{
   alert( “Hello, John!” );
}
```



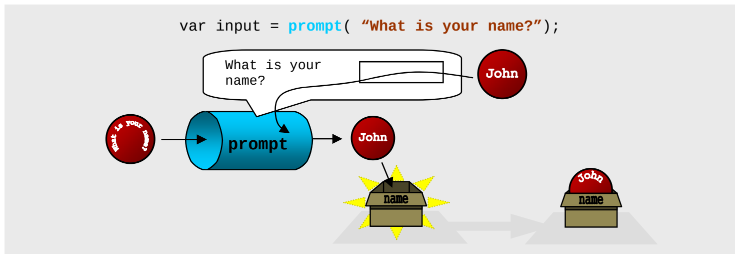




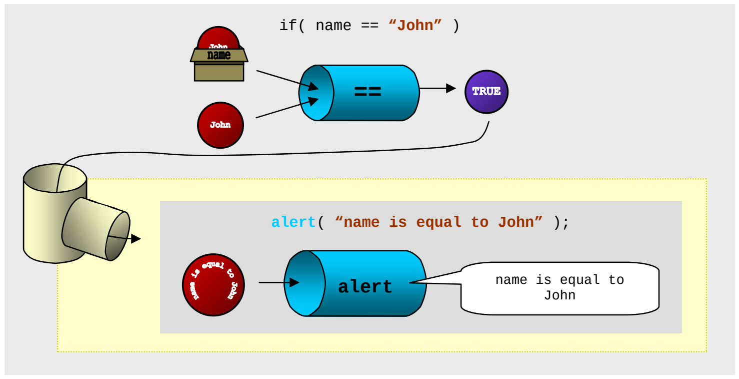
Now try running it again, but this time enter **"John"**.

In addition to simply running some code if a condition is true, there is also a way to set a default block of code if that same condition is false. We do this with the **else** keyword.

We'll now add an **else** to our previous script. The conditional can now be read as "**if name is equal to "John", run some code, else run some other code**".

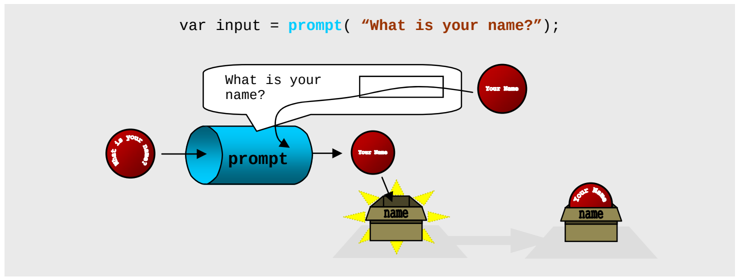


```
var input = prompt( "What is your name?");
```



```
if( name == "John" )
```

```
alert( "name is equal to John" );
```

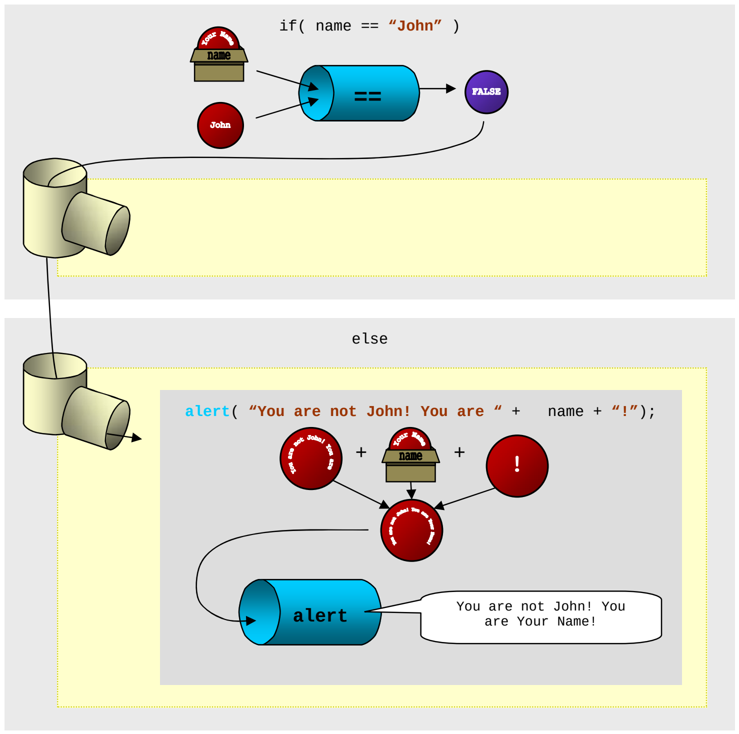


```
var input = prompt( "What is your name?");
```

The first run through of this script, enter your own name. Assuming your name is not John, the Boolean will fall through and the code inside the **else** block will run.

```
var name = prompt( “What is your
   name?” );
if( name == “John” )
{
   alert( “Hello, John!” );
}
else
{
   alert( “You are not John! You are “ +
        name + “!” );
}
```

Now, just like before, run the script again. This time, enter “**John**” instead of your name. The **if** statement will catch the Boolean and the **else** block will be ignored.

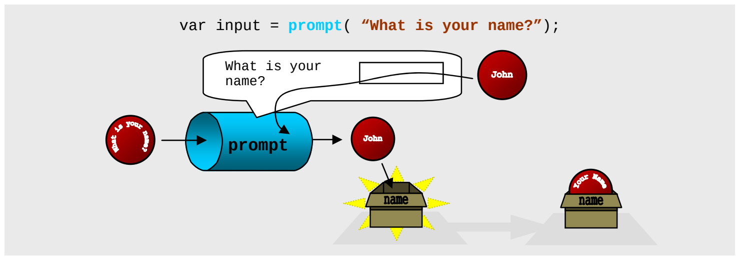


```
if( name == “John” )
```

```
else
```

```
alert( “You are not John! You are “ +   name + “!”);
```

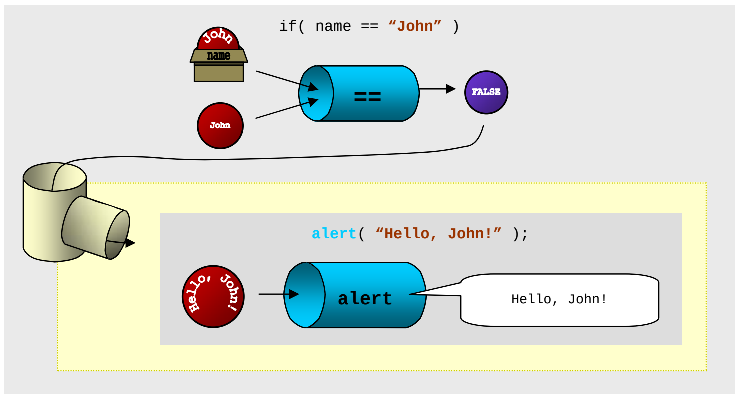
```
You are not John! You
  are Your Name!
```

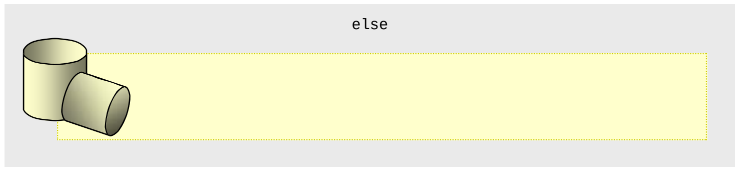


```
var input = prompt( “What is your name?”);
```

```
What is your
name?
```

This block is intentionally left empty, because the code inside the **else** block will not run.





Conclusion

All good things must come to an end and it looks like we've run out of time for today. While these basic concepts are fundamental to programming, they barely scratch the surface.

We've learned about basic objects, like strings and numbers. We've learned about using and making functions. We've learned about Boolean objects and comparisons. We've learned about conditional statements. We've learned a lot.

There is a lot more left to learn, though. There are a lot of other fundamental concepts that are a little more complex, such as **loops** and **arrays**. Because they are more advanced, these concepts are not for absolute beginners, and thus are not covered in this book.

Fortunately for you, however, you are no longer an absolute beginner.

With a little effort, it shouldn't be long before you master these and many other concepts. Of course, if a little effort is too much effort for your tastes, you could always look out for "**An Illustrated Guide to Programming for Advanced Beginners and the Not-So-Computer Illiterate**".

Thanks for reading.