Comparisons and Decisions

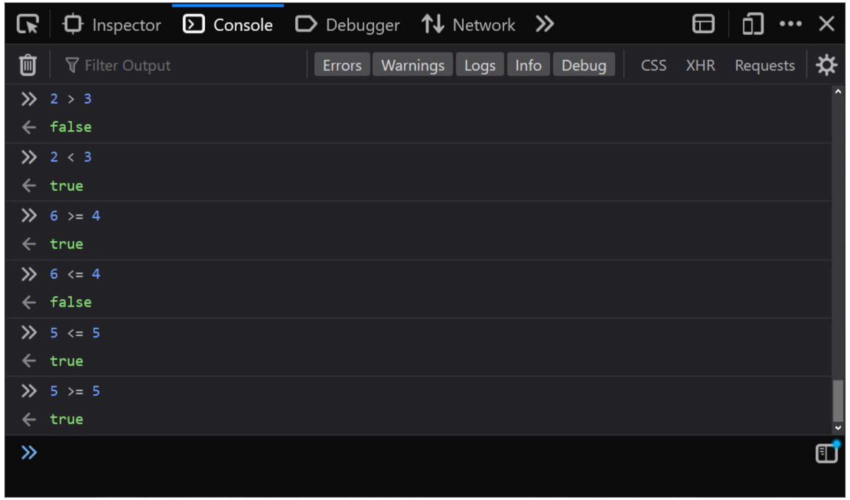
JavaScript

Comparison Operators

We can compare numerical values, to see if one is greater or less than the other, using [relational operators](). Give them a try in the Web Console!

These expressions will return to you either true or false, depending on if the statement is true or false.

<	Less than.
>	Greater than.
<=	Less than or equal to.
>=	Greater than or equal to.



```
» 2 > 3
← false
» 2 < 3
← true
» 6 >= 4
← true
» 6 <= 4
← false
» 5 <= 5
← true
» 5 >= 5
← true
»
```

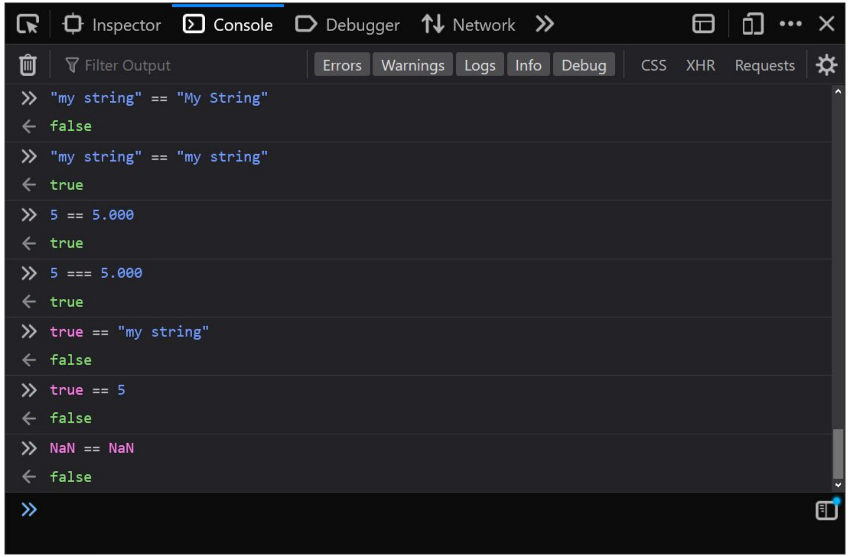
Equality Operators

Equality operators can be a little more tricky, as JavaScript is loose with data-types.

Run lots of experiments to see which expressions evaluate to true or false!

== Equality operator, checks if the values on either side appear to be equivalent.

=== Identity operator, is similar but always considers values of different types to be different.



```
>> "my string" == "My String"
<- false
>> "my string" == "my string"
<- true
>> 5 == 5.000
<- true
>> 5 === 5.000
<- true
>> true == "my string"
<- false
>> true == 5
<- false
>> NaN == NaN
<- false
>>
```

Logical Operators

Logical operators are used for comparing two results.

Regardless of the two values being compared, they'll be treated as the nearest boolean equivalent (true or false.)

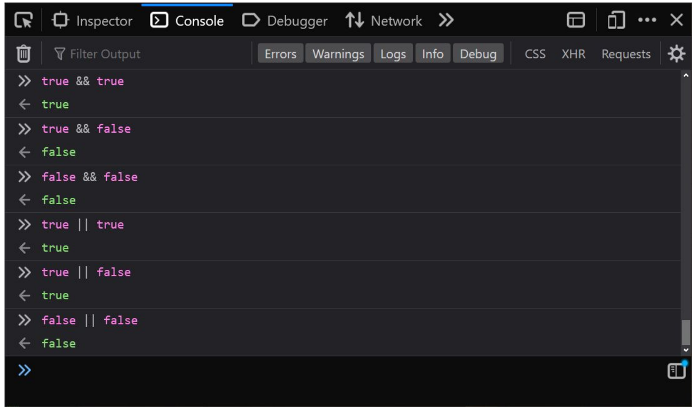
The first few times you use logical operators, they can be a bit of a thought exercise. Later on, they can get a bit complex too as you can chain them!

Try to keep logical operator comparisons grouped within parentheses to make these checks a bit more obvious when you can.

The logical operators you'll see most often include…

&& And, means that the result on both sides must be true.

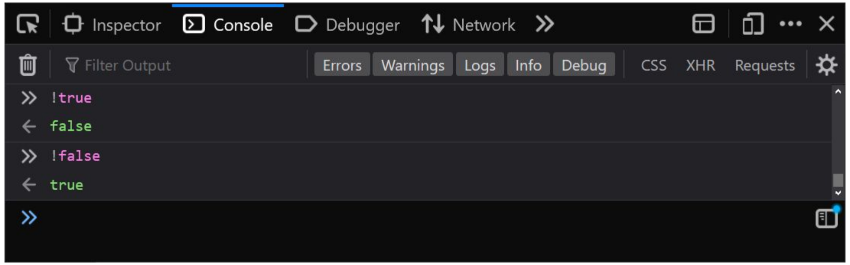
|| Or, means that a result on either side can be true.



```
>> true && true
<- true
>> true && false
<- false
>> false && false
<- false
>> true || true
<- true
>> true || false
<- true
>> false || false
<- false
>>
```

Not

If you'd like to treat an evaluation as its boolean opposite, you can place an exclamation mark before it.



```
>> !true
← false
>> !false
← true
>>
```

If Statement

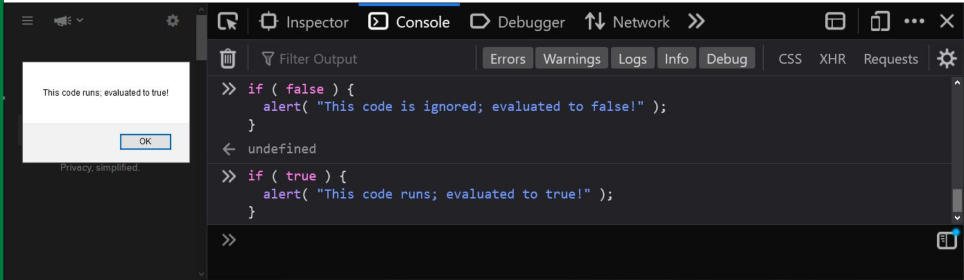
When our programs need to make decisions, the usual go-to is the "if" statement.

If statements pose a statement that is tested for a true or false result.

If the statement is true, please execute (run) the following code block.

The syntax involves…

- The if keyword.
- Parentheses containing an expression to evaluate as either true or false.
- A code block; they are marked by curly braces "{}". The lines of code inside are considered to be inside of that code block.



This code runs; evaluated to true!

```
>> if ( false ) {
     alert( "This code is ignored; evaluated to false!" );
   }
<- undefined
>> if ( true ) {
     alert( "This code runs; evaluated to true!" );
   }
>>
```

Else Statement

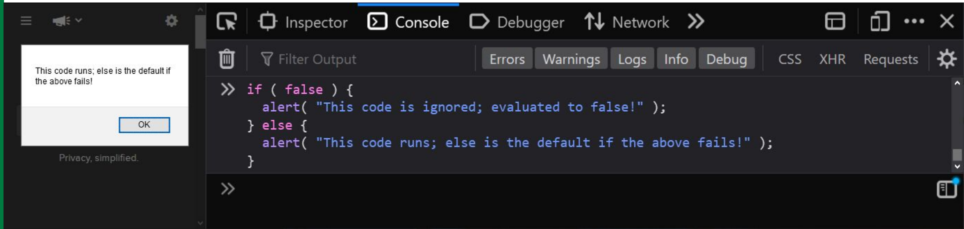
Suppose you need something to happen if the evaluation was true… but if it isn’t—you need an alternative set of instructions to run instead! This is where “else” comes in.

When you want one of two sets of instructions to be the only possibilities at that point in the program, you can write two “if” statements.

However, this is more difficult to maintain and harder to read than one of the available alternative solutions. Enter: the “else” statement!

Else is a default set of instructions that takes place only if your “if” condition evaluates to false. Think of it as a fallback.

Note: you cannot add an expression to test in parentheses for an else statement. An else must immediately follow an if code block.



This code runs; else is the default if the above fails!

```
>> if ( false ) {
     alert( "This code is ignored; evaluated to false!" );
   } else {
     alert( "This code runs; else is the default if the above fails!" );
   }
```

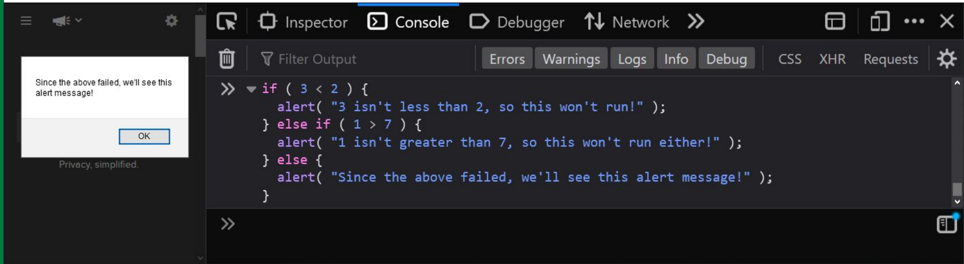
Else If Statement

If you need to decide between more than two possibilities, you can add as many “else if” statements if you’d like between an if code block and an else statement.

“else if” statements are placed in between an if statement and else statement. You can chain many together if you need to!

They use the keywords “else if”, followed by parentheses containing an expression, and a code block with instructions.

Try alternative expressions, and see if you can get different statements to run or be ignored!



Since the above failed, we'll see this alert message!

```
if ( 3 < 2 ) {
  alert( "3 isn't less than 2, so this won't run!" );
} else if ( 1 > 7 ) {
  alert( "1 isn't greater than 7, so this won't run either!" );
} else {
  alert( "Since the above failed, we'll see this alert message!" );
}
```