

Introduction to Javascript

HTML = to define the content of web pages

CSS = to specify the layout of web pages

JavaScript = to program the behavior of web pages

A computer program is a list of "instructions" to be "executed" by a computer. In a programming language, these programming instructions are called statements. A JavaScript program is a list of programming statements.

JavaScript statements are composed of:
Values, Operators, Expressions, Keywords, and
Comments.

Loading Javascript

It is a best practice to have a separate file for your javascript. Such a file ends with the abbreviation `.js`, you can for example call your file `script.js`

This file should be linked in the head of your html file. the `<script>` link looks like this:

```
<script type="text/javascript"
src="script.js"></script>
```

Values

In JavaScript's system, most data is neatly separated into things called values. Every value has a type, which determines the kind of role it can play. There are six basic types of values: Numbers, strings, booleans, objects, functions, and undefined values.

Data Types

Let's first go over the following data types:

- numbers
- strings
- booleans
- arrays

Numbers

JavaScript has only one type of number. Numbers can be written with or without decimals.

```
var x = 3.14;  
var y = 3;
```

Strings

Strings are used for storing textual information. It can be as short as a word or as long as a sentence. To define a string, use double or single quotes:

```
'This is textual information within single quotes'  
"This is textual information within double quotes"
```

Boolean Values

It is often useful to have a value that distinguishes between only two possibilities. For this purpose, JavaScript has a Boolean type, which has just two values: `true`, or `false`. Think of a boolean as an on/off or a yes/no switch.

Arrays

JavaScript provides a data type specifically for storing sequences of values. It is called an array and is written as a list of values between square brackets, separated by commas.

Array Index

An empty array looks like this: `[]`

In it we can place a string element with quotation marks:

`["one"]`

When adding more elements they have to be separated by a comma: `["first", 2, "three"]`

You can refer to an element in the array by referring to the index number. The notation for getting at the elements inside an array also uses square brackets `[]`.

It is important to know that Javascript is a zero indexed language, meaning it starts counting from 0 instead of 1.

Below shows how each element in the array can be selected:

`["red", "white", "blue"][0]` selects "red"

`["red", "white", "blue"][1]` selects "white"

`["red", "white", "blue"][2]` selects "blue"

The `.length` property of an array tells us how many elements it has. This property name is a valid binding name, and we know its name in advance, so to find the length of an array, you write `[].length`

In the following array this would give us a length of 3:

```
["red", "white", "blue"].length
```

Variables

All these values can be used in Javascript but will dissipate after being used because they are not bound to anything.

In order to catch or hold these values, JavaScript provides a thing called a binding, or variable. You can use the **let** keyword to declare variables. It is followed by the name we gave the variable and then an equal sign **=** is used to assign values to the variable.

A variable can first be declared: `let name;`

And then get assigned a value: `name = 'Jacob';`

This can also be done at once: `let name = 'Jacob';`

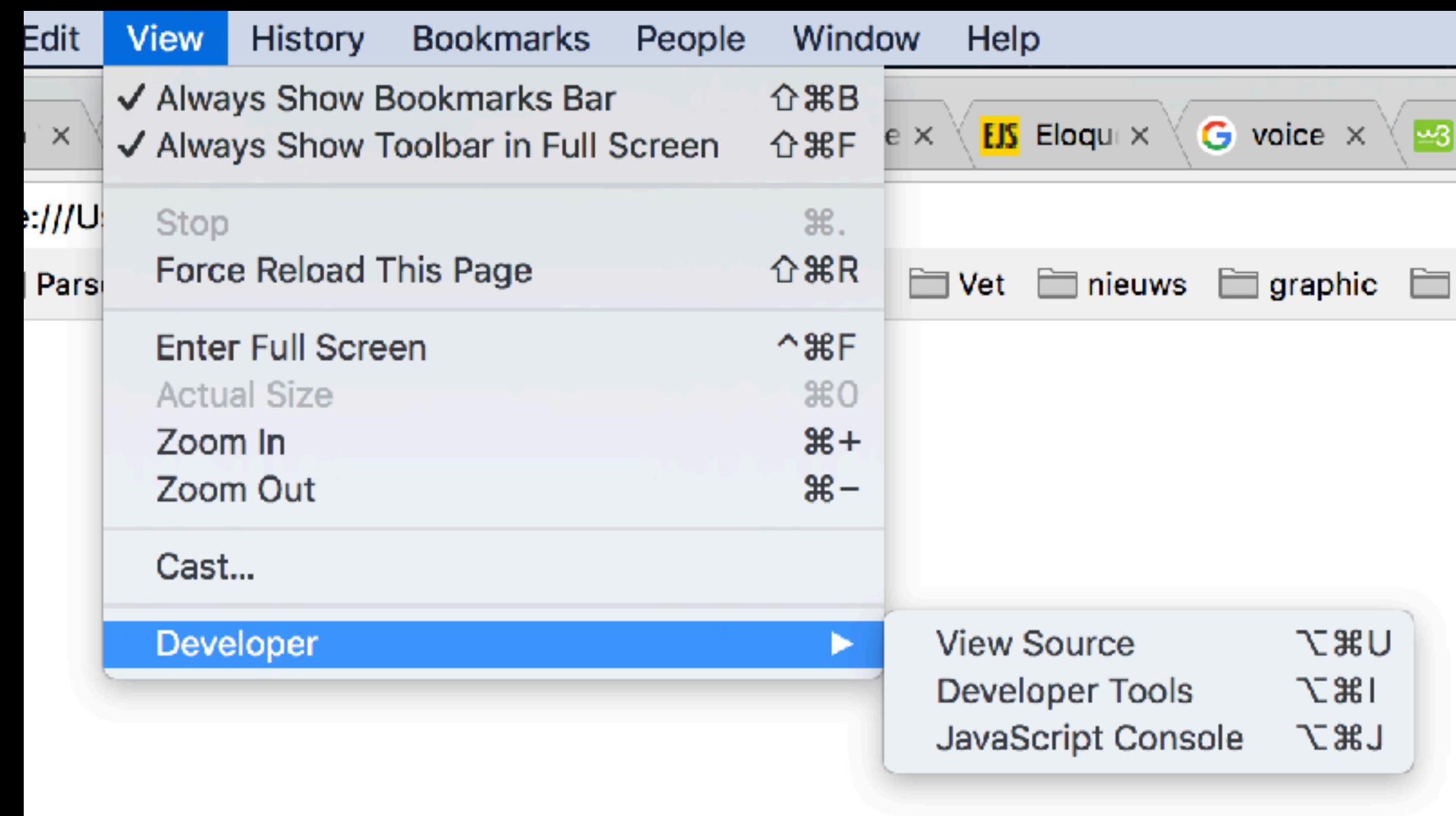
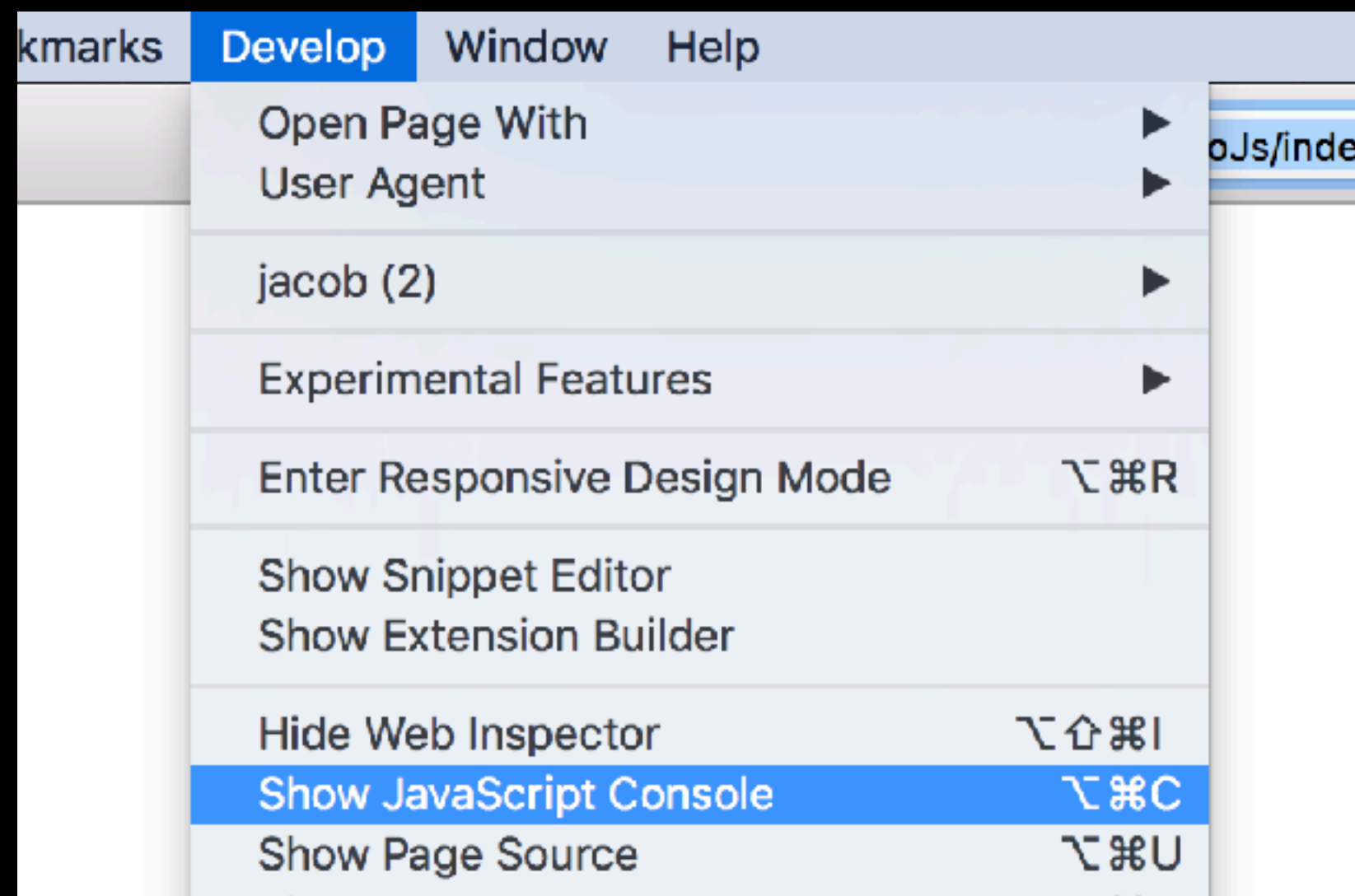
A variable can be reassigned a different value at any time:

`name = 'Jacob Hoving';`

Console Log

For the purpose of this tutorial we will only write into our browser's console.

Here's how to open up the console in your browser:
safari: Develop > Show Javascript Console
Chrome: View > Developer > Javascript Console



Now let's try logging something into our console:

```
console.log("Hello world");
```

Console log accepts any variable or data type. Multiple values can be strung together, to do this they have separated with a comma:

```
console.log(myLet, "Hello", 7);
```

Operators

We already used the `=` operator to assign a value to a variable. This is an assignment operator. JavaScript operators are used to assign values, compare values, perform arithmetic operations, and more.

Arithmetic Operators

Arithmetic Operators can be used to do math in javascript.

Addition

1 + 1 will give 2

Subtraction

3 - 2 will give 1

Multiplication

5 * 3 will give 15

Division

10 / 2 will give 5

```
console.log(1 + 1);  
console.log(3 - 1);  
console.log(5 * 3);  
console.log(10 / 2);
```

2

2

15

5



Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

== is Equal to

!= is not Equal

< Less than

> Greater than

<= Less than or equal to

>= Greater than or equal to

Below are some comparisons:

`1 < 2` is true

`2 <= 1` is false

`1 <= 1` is true

`1 > 1` is false

```
var x = 4,  
    y = 6;
```

`x < y` is true

```
console.log(1 < 2);  
console.log(2 <= 1);  
console.log(1 <= 1);  
console.log(1 > 1);  
var x = 4,  
    y = 6;  
console.log(x < y);
```

true

false

true

false

true

The following comparisons check if the values are equal:

`10 === 10;` is true

`10 === 5;` is false

`"hi" === "hi";` is true

`"Hi" === "hi";` is false

```
console.log(10 === 10);  
console.log(10 === 5);  
console.log('hi' === 'hi');  
console.log('Hi' === 'hi');
```

true

false

true

false

Logical Operators

Logical operators are used to determine the logic between variables or values.

&& stands for “and”

(5 < 10 && 5 > 1) is true

|| stands for “or”

(5 === 5 || 4 === 5) is true

! stands for “not”

!(x === y) is true

```
console.log(5 < 10 && 5 > 1);  
console.log(5 === 5 || 4 === 5);  
console.log(!(4 === 5));
```

true

true

true



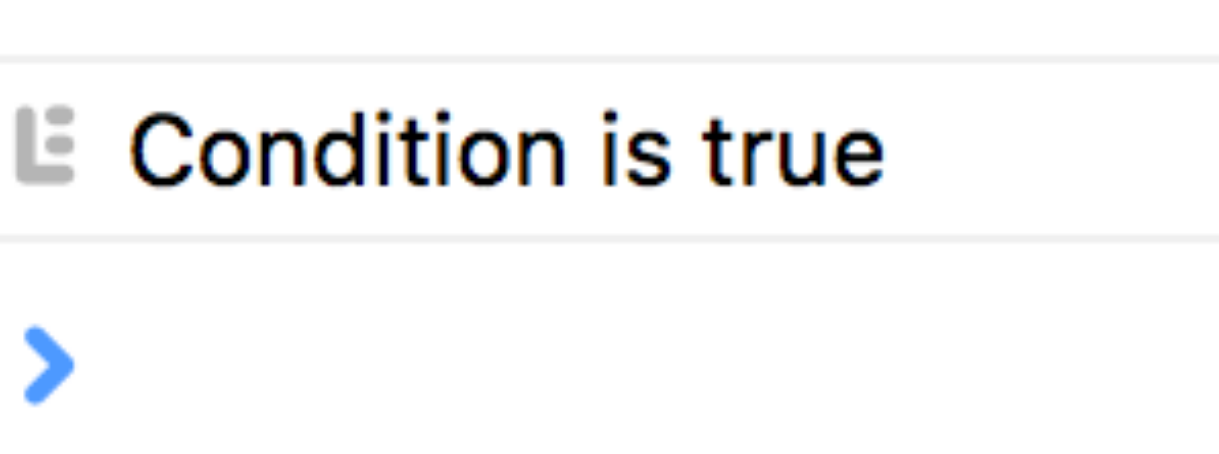
If and Else

Not all programs are straight roads. We may, for example, want to create a branching road, where the program takes the proper branch based on the situation at hand. This is called conditional execution.

Conditional execution is created with the `if` keyword in JavaScript. In the simple case, we want some code to be executed if, and only if, a certain condition holds.

When the if condition is true it will execute:

```
if (true) {  
    console.log('Condition is true');  
}
```

A screenshot of a browser's developer console. It shows a single log entry with the text "Condition is true". To the left of the text is a small icon representing a log entry, and to the right is a blue arrow pointing to the right, indicating it's the current entry.

Condition is true

When the if condition is false it won't execute:

```
if (false) {  
    console.log('This will never execute');  
}
```


You often won't just have code that executes when a condition holds `true`, but also code that handles the other case. The `else` keyword can be used, together with `if`, to create two separate, alternative execution paths.

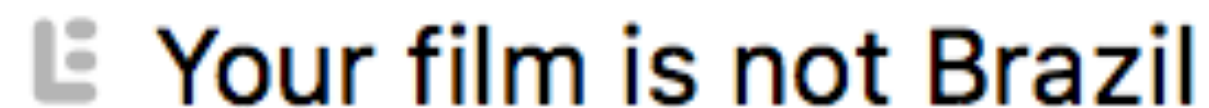
When the if condition is false the else condition will execute:

```
let film = 'Bladerunner';
```

```
if (film === 'Brazil') {  
  console.log('Your film is Brazil.');
```

```
} else {  
  console.log('Your film is not Brazil');
```

```
}
```



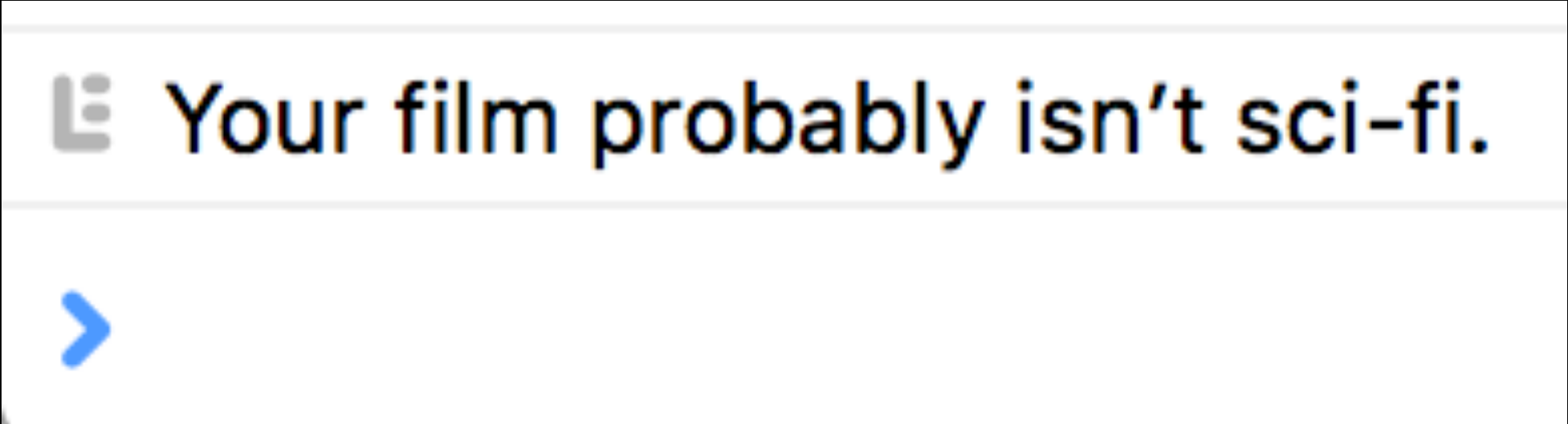
⌵ Your film is not Brazil



It is also possible to add more checks by adding a value separated by a logical operator.

```
let film = 'The Gleaners and I';
```

```
if (film === 'Bladerunner' || film === 'Brazil') {  
  console.log('Your film is sci-fi');  
} else {  
  console.log('Your film probably isn't sci-fi.');}
```



The screenshot shows a browser's developer console. At the top, there is a light gray header bar with a small icon on the left and the text "Your film probably isn't sci-fi." in a dark font. Below this header is a white area containing a blue chevron icon pointing to the right.

▮ Your film probably isn't sci-fi.



For Loop

Loops can execute a block of code a number of times.

Loops are handy, if you want to run the same code over and over again, each time with a different value. Often this is the case when working with arrays.

Especially when looping, a program often needs to “update” a binding to hold a value based on that binding’s previous value.

This can be done as follows: `myLet = myLet + 1;`

JavaScript provides a shortcut by using the increment operator: `myLet++;`

The same goes for: `myLet = myLet - 1;`

Javascript provides a shortcut by using the decrement operator: `myLet--;`

The **for** loop is often the tool you will use when you want to create a loop.

The parentheses **()** after a **for** keyword must contain two semicolons**;**. The part before the first semicolon**;** initializes the loop, usually by defining a variable. The second part is the expression that checks whether the loop must continue. The final part updates the state of the loop after every iteration.

The loop keeps entering that statement within the brackets as long as the expression produces a value that gives **true** when converted to Boolean.

```
for (let multiplier = 1; multiplier <= 10; multiplier++) {  
    let result = multiplier * 4;  
    console.log(result);  
}
```

Add the following loop and see what it will produce in your console log:

```
for (let multiplier = 1; multiplier  
<= 10; multiplier++) {  
  let result = multiplier * 4;  
  console.log(result);  
}
```

4

8

12

16

20

24

28

32

36

40

Loops can also use already existing variables. In the example below the loop is used to log all the separate elements within the array:

```
let array = [1, 2, 3];
```

```
for (let counter = 0; counter < array.length;  
    counter++) {  
    console.log(array[counter]);  
}
```

```
let array = [1, 2, 3];
```

```
for (let counter = 0; counter < array.length; counter++) {  
    console.log(array[counter]);  
}
```

1

2

3

>

A loop can also contain another loop. In the example below another loop is added to the previous loop. Play around with this loop the remaining time of the class to see what you can change about it.

```
console.log('hello world');
```

```
for (let multiplier = 1; multiplier < 11;
multiplier++) {
  console.log(multiplier * 6);
  let star = '*';
  for (let i = 0; i < 20; i++) {
    star = star + '*';
    console.log(star);
  }
}
```

To get a good sense about javascript I recommend you to go through more online tutorials and really dig into them.

A good places to start is:

<http://eloquentjavascript.net/>

or

<https://www.w3schools.com/js/default.asp>

or

<https://www.codecademy.com/learn/introduction-to-javascript>