Design and Implementation of Online Behavioral Experiments

nodeGame.org

Stefano Balietti

MZES and Heidelberg

Introduction to JavaScript

@balietti
@nodegameorg
stefanobalietti.com@gmail.com

JavaScript

- JavaScript was developed in May 1995 by *Brendan Eich* for Netscape Communications Corp
- Was created in **10 days** in order to accommodate the Navigator 2.0 Beta release
- Initially called **Mocha**, later renamed **LiveScript** in September, and later **JavaScript** in the same month

JavaScript

- Microsoft introduced **JScript** as reverse-engineered implementation of Netscape's JavaScript in 1996 in Internet Explorer 3

- In 1996 Netscape submitted JavaScript to European Computer Manufacturers Association (ECMA) to create and industry standard

- In 1997 **ECMAScript** was released

- Between 1997 and 2009 5 standard have been released.

- *July 2015 ECMASCRIPT V6 released.*

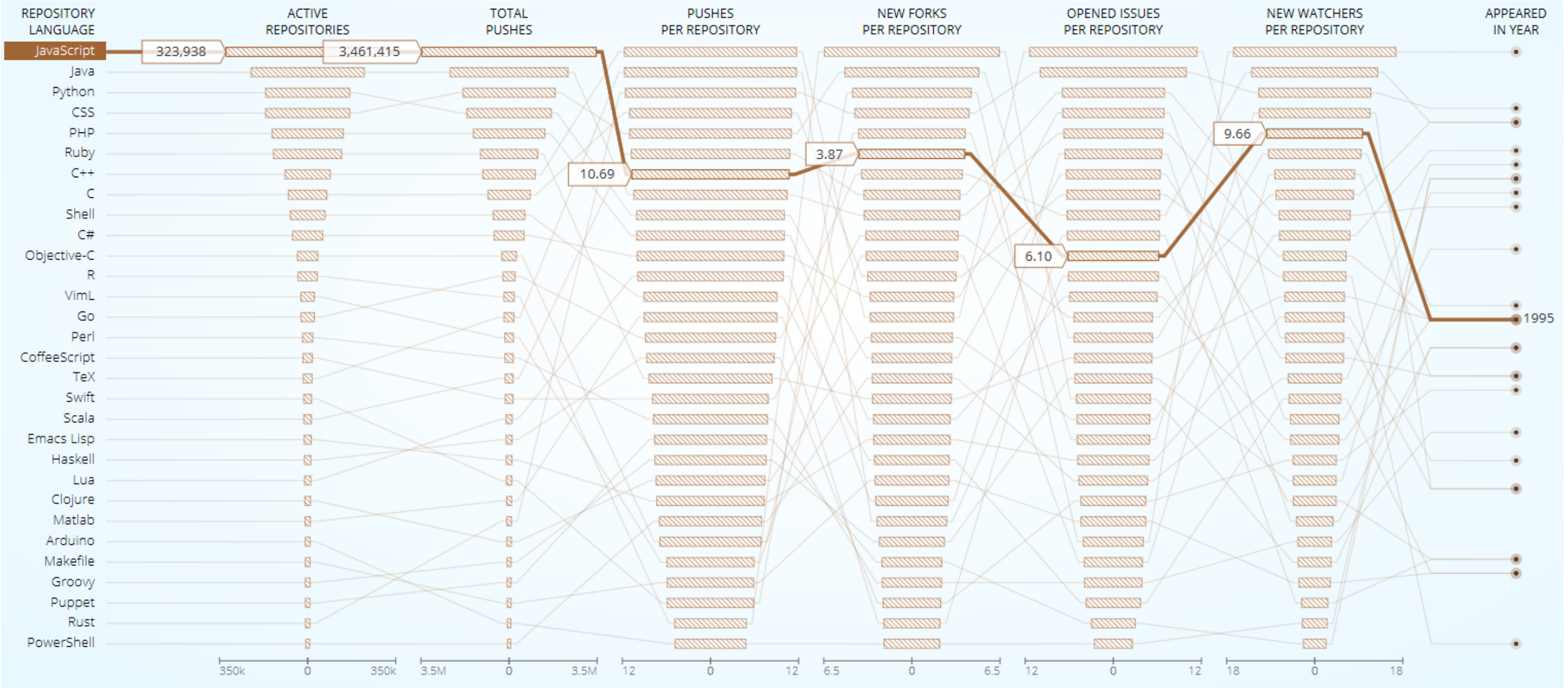
Node.JS

- **Node.JS** was invented in 2009 by *Ryan Dahl* and other developers working at Joyent

- Combination of Google's V8 JavaScript engine, an event loop, and a low-level I/O API

- **npm**, the node package manager, in 2011

- Versions: 0.10, 0.12, 4.0 … 10.0!



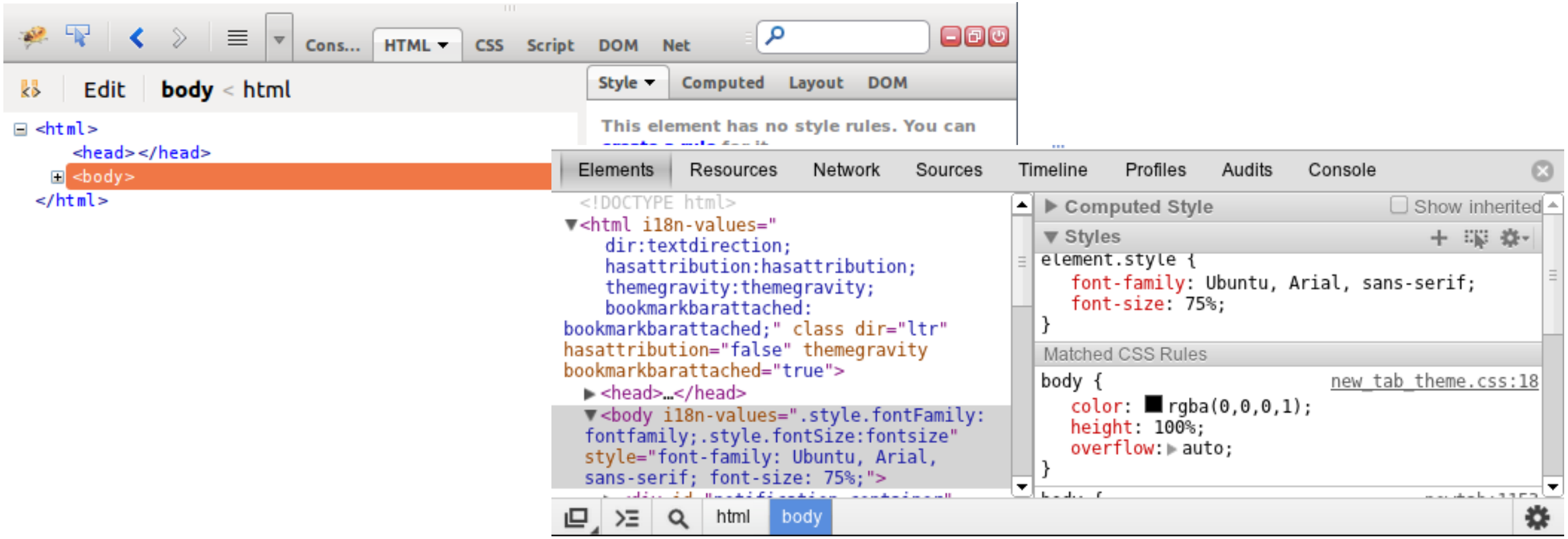
JavaScript is #1 Language on Github

http://githut.info

Hands On 1

Open the JavaScript console of your browser:

(ctrl+shift+I or Right Click/Inspect Element)



Hands On 1

Try to open different web sites, how does the content of the console changes?

Hands On 1

bahn.de: lots of messy output, including your first and last name

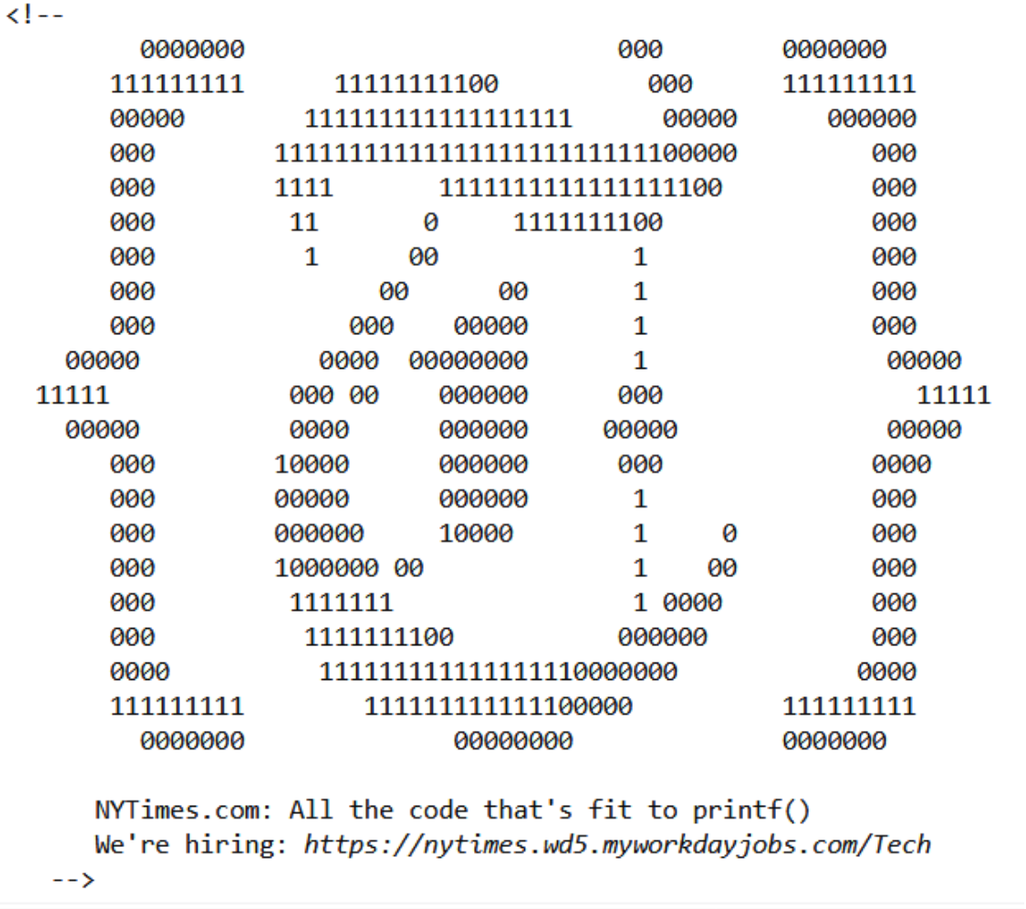
```
▼ Incoming message 'load'                                                    common.js:46:37
  ▼ load(Kunde)                                                              common.js:46:37
    ⓘ [iLogic] Connectivity is  Connected                                    common.js:137:46
      [Cache] Read  Kunde  from cache (storage):  { "khash" :                common.js:114:51
      "48fef9ed13945b104be7f743779d182f76de070da[redacted]" , "name" : { "nachname" : "Balietti"
      , "vorname" : "Stefano" , "anrede" : "0" , "titel" : "1" , "login" : "[redacted]}}
    ⓘ [iLogic] Data is in cache but outdated/expired.                        common.js:137:46
    ⓘ [iLogic] -> loading it from server.                                    common.js:137:46
    ⓘ [iLogic] Ajax call load(Kunde).                                        common.js:137:46
▼ Processing AJAX response for load(Kunde)                                   common.js:46:37
      [iLogic]   response =                                                  common.js:114:51
      ▶ Object { status: 200, content: "{ \"khash\" :
      \"48fef9ed13945b104be7f743779d182f76de070da[redacted]\" , \"name\" : { \"nachname\" :
      \"Balietti\" , \"vorname\" : \"Stefano\" , \"anrede\" : \"0\" , \"titel\" : \"1\" , \"login\" :
      \[redacted]\"}}", etag: "jfu-ET6GJjDvpkdIkd8[redacted]" }
      [Cache] Wrote  Kunde  to cache (storage):  { "khash" :                 common.js:114:51
      "48fef9ed13945b104be7f743779d182f76de070da[redacted]" , "name" : { "nachname" : "Balietti" ,
      "vorname" : "Stefano" , "anrede" : "0" , "titel" : "1" , "login" : [redacted]}}
```

Hands On 1

facebook.com: a warning to not fall victim of social engineering phishing attacks

```
 .d8888b.  888                            888
d88P  Y88b 888                            888
Y88b.      888                            888    This is a browser feature intended for
 "Y888b.   888888  .d88b.  88888b.   888    888    developers. If someone told you to copy
    "Y88b. 888    d88""88b 888 "88b  888    888    and paste something here to enable a
      "888 888    888  888 888  888  888    Y8P    Facebook feature or "hack" someone's
Y88b  d88P Y88b.  Y88..88P 888 d88P            account, it is a scam and will give them
 "Y8888P"   "Y888  "Y88P"  88888P"   888    access to your Facebook account.
                           888
                           888
                           888

See https://www.facebook.com/selfxss for more information.
```

Hands On 1

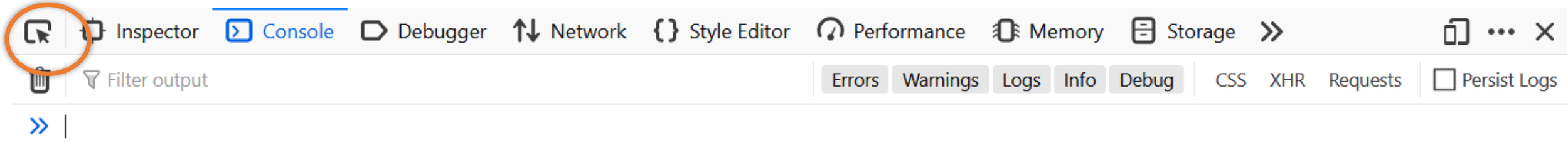


nytimes.com: a job offer!

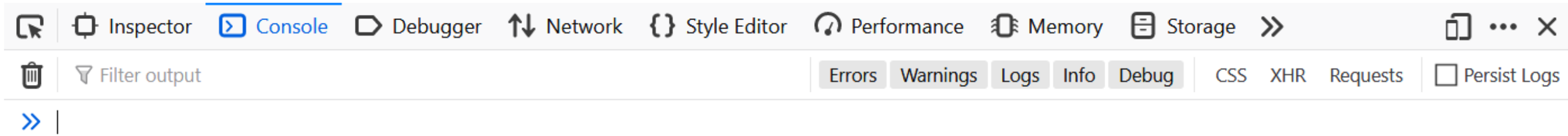
Hands On 1

Try to open different web sites, how does the content of the console changes?

Clear any pre-existing output: click on button or type `clear()`



Hands On 1

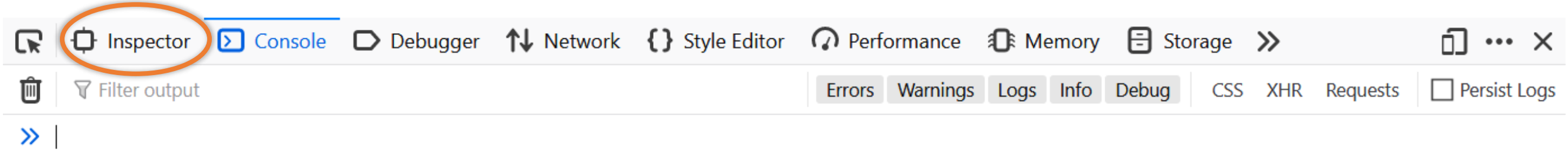


Type something in the console using the command:

```
console.log('This is my very own text');
```

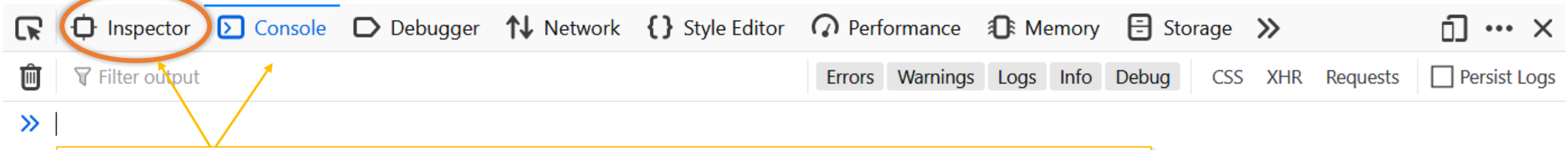
Hands On 1

The Inspector is where you can visualize the live DOM (Document Object Model) and make changes, including CSS (Cascading Style Sheets) changes.



Hands On 1

The Inspector is where you can visualize the live DOM (Document Object Model) and make changes, including CSS (Cascading Style Sheets) changes.



The actual names might be slightly different depending on the browser version and language. For instance, "Inspector" is sometimes called "Elements."

Some of the HTML Page's Inhabitants

DOM Tree

```
<HTML>
   <HEAD>
      <LINK>
      <SCRIPT>
   </HEAD>
   <BODY>
   …
   </BODY>
</HTML>
```

Presentation Tags

<P>
<DIV>
<SPAN>

Images and Links

<IMG>
<A>

Forms

<INPUT>
<TEXTAREA>

Attributes

<DIV id="header">
<SPAN class="bold">
<IMG SRC="image.jpg" />
<A HREF="newpage.htm">
<INPUT disabled>

CSS Declarations

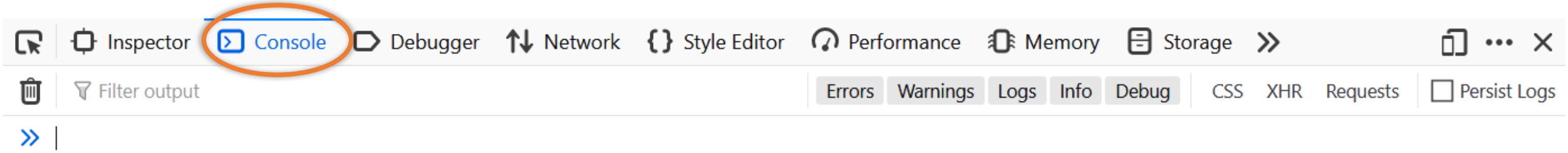
.bold { font-weight: bold };
#header { width: 600px };

Hands On 1: Browser's Dev Tools

Read what you download.

Hands On 1: Browser's Dev Tools

The console is where you can type JavaScript code to interact with the page



Hands On 1: Browser's Dev Tools

Changes in the Inspector are immediately reflected on the page.

For example, if add a rule:

"display: none"

the selected element will be hidden in the page.

JavaScript

Great tutorial from novice to JavaScript Ninja:
http://javascript.info/

Variables

Variables

http://javascript.info/

Variables

```
var message = 'Hello!';
```

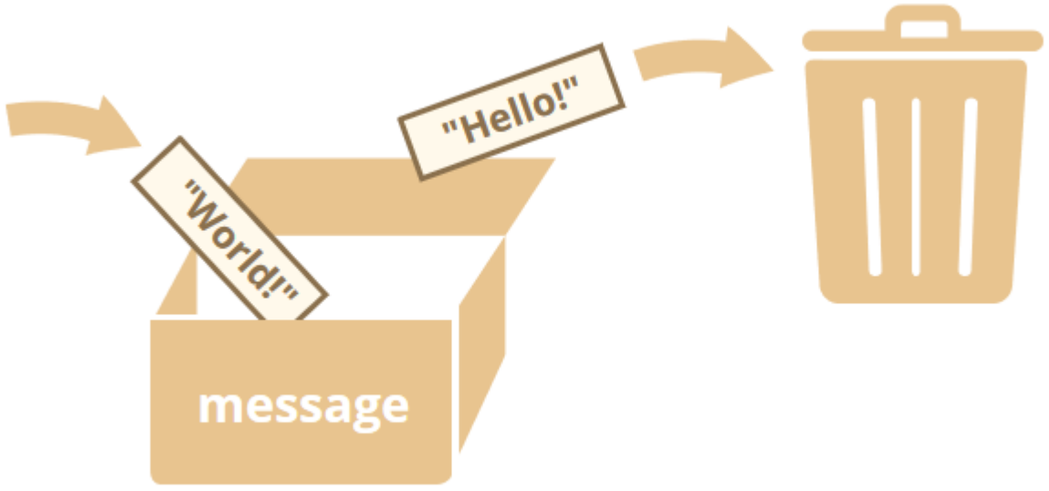
Variables

http://javascript.info/

Variables

```
var message = 'Hello!';
```


```
// value changed.
message = 'World!';
```




```
alert(message);
```

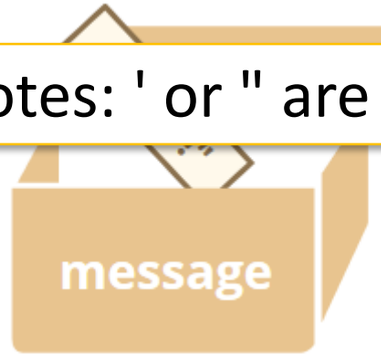
Variables

http://javascript.info/

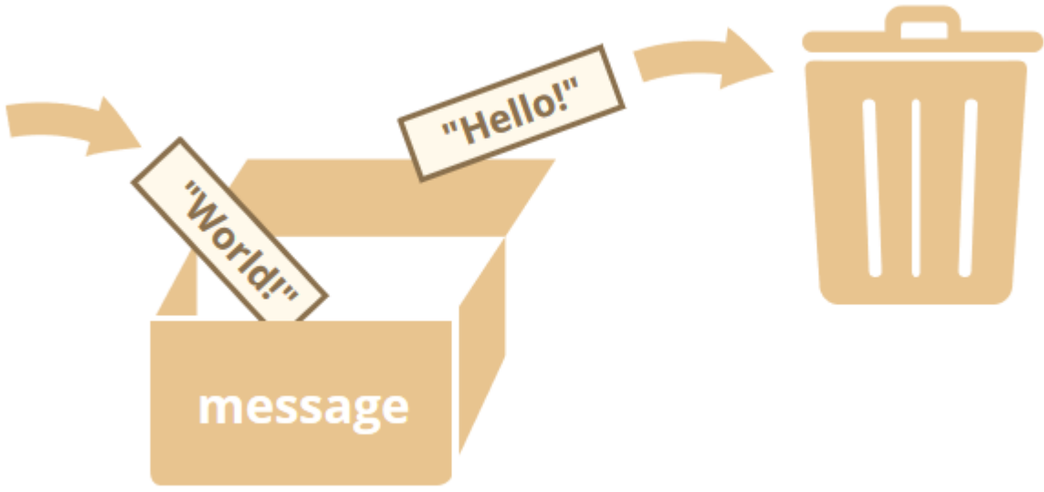
Variables

Strings must be wrapped in quotes: ' or " are equivalent.

```
var message = 'Hello!';
```




```
// value changed.
message = 'World!';
```




```
alert(message);
```

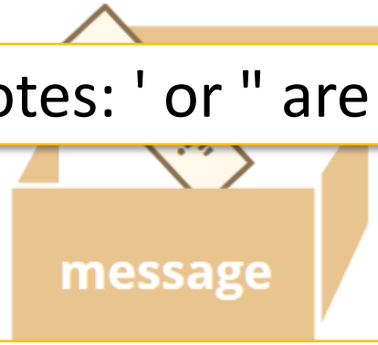
Variables

http://javascript.info/

Variables

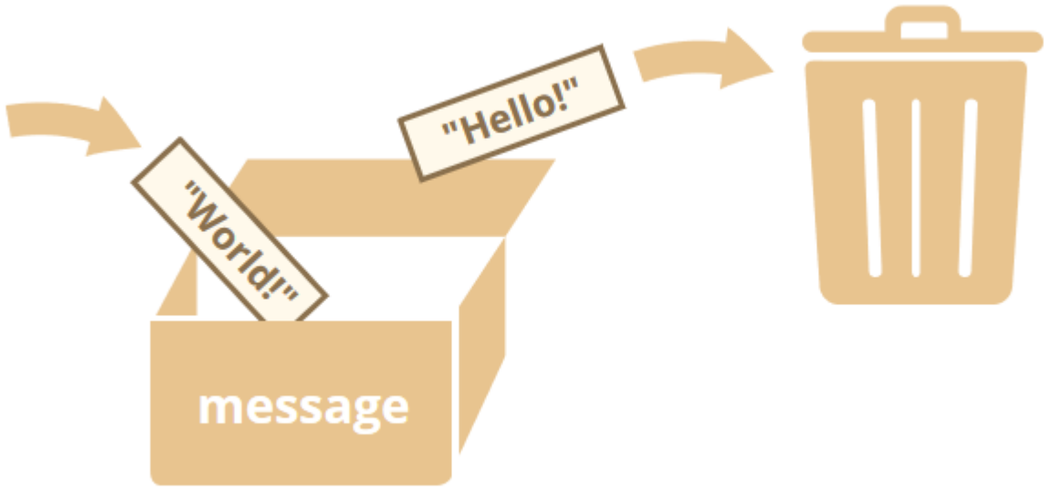
Strings must be wrapped in quotes: ' or " are equivalent.

```
var message = 'Hello!';
```



Text following // is a comment and it is not read by JavaScript

```
// value changed.
message = 'World!';


alert(message);
```



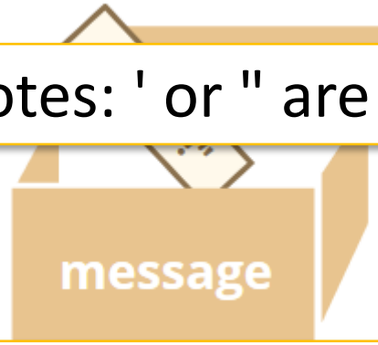
Variables

http://javascript.info/

Variables

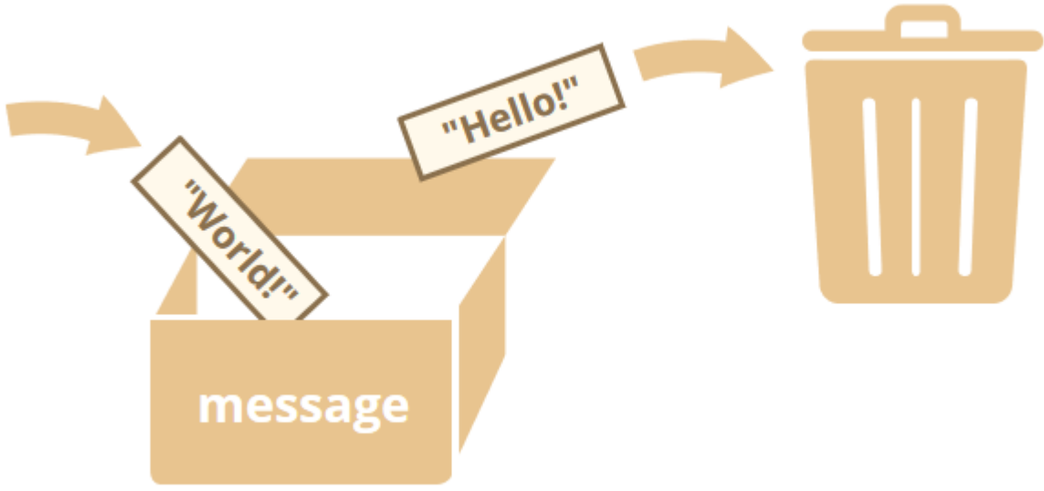
Strings must be wrapped in quotes: ' or " are equivalent.

```
var message = 'Hello!';
```



Text following // is a comment and it is not read by JavaScript

```
// value changed.
message = 'World!';
```



Displays the content in a popup window

```
alert(message);
```

Variables

```
var message = 'Hello!';
// value changed.
message = 'World!';
alert(message);

// type of value changed to number
message = 2019;

// string concatenation (works also with numbers).
alert('This is year ' + message);
```

Variables

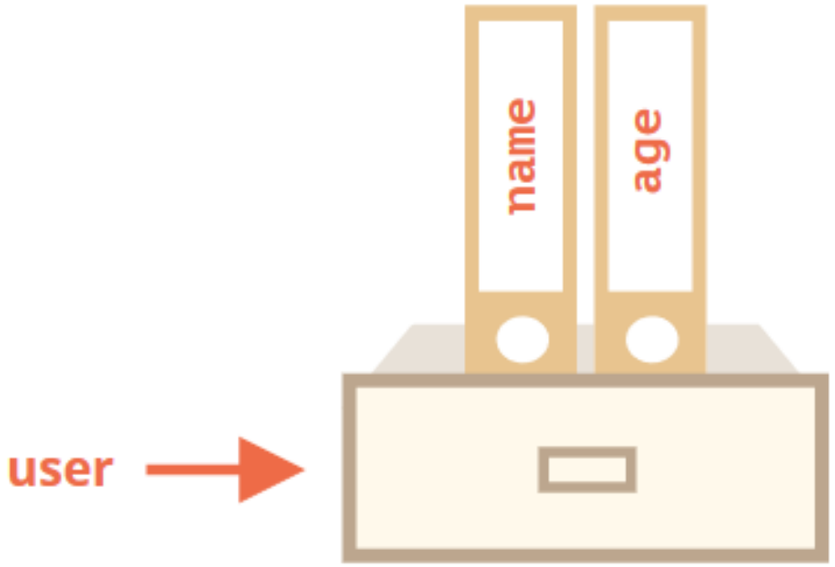
```
var message = 'Hello!';
// value changed.
message = 'World!';
alert(message);

// type of value changed to number
message = 2019;

// string concatenation (works also with numbers).
alert('This is year ' + message);
```

Variable are "loosely typed," that is their type (string, number, etc.) can be changed after assignment

Objects

http://javascript.info/

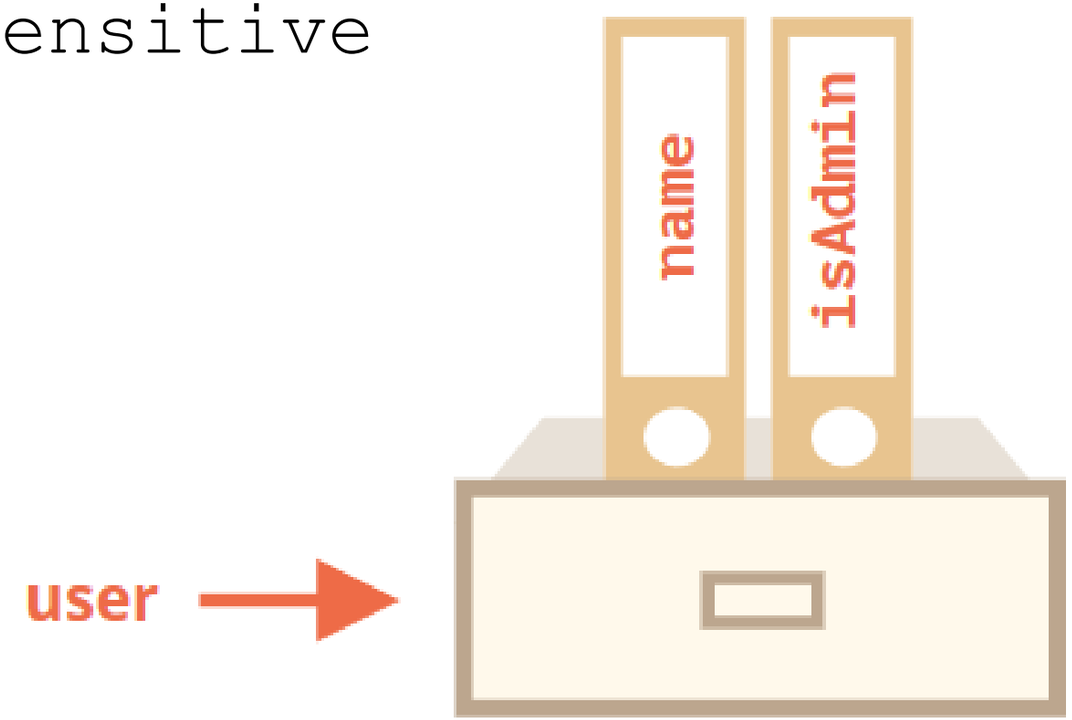
```
var user = {
  name: "John", // by key "name" store value "John"
  age: 30       // by key "age" store value 30
};
```



Objects

http://javascript.info/

```
// We now add a new property
// Note! JavaScript is case sensitive
user.isAdmin = true;
// Delete an existing one.
delete user.age;
```

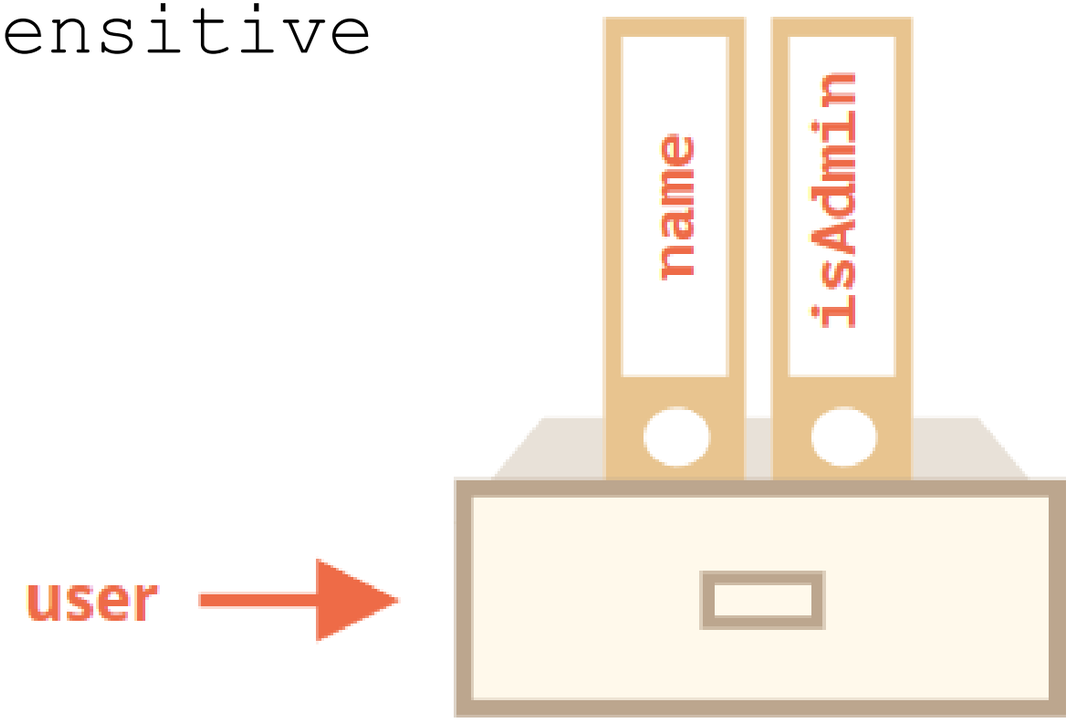


Objects

http://javascript.info/

```
// We now add a new property
// Note! JavaScript is case sensitive
user.isAdmin = true;
// Delete an existing one.
delete user.age;
```

The dot operator accesses the value of a given property inside the object.
If the property was not previously defined (as in this case), it will be simply created.



Looping in Objects (For In)

```
for (var property in user) {
  if (user.hasOwnProperty(property)) {
    console.log(property + ': ' + user[property]);
  }
}

// Output.
// name: John
// isAdmin: true
```

- **hasOwnProperty** is necessary to avoid contamination of other properties belonging to the object and not added by the user
- **MUST USE ALWAYS.**

Looping in Objects (For In)

```
for (var property in user) {
  if (user.hasOwnProperty(property)) {
    console.log(property + ': ' + user[property]);
  }
}

// Output.
// name: John
// isAdmin: true
```

- The square parentheses allows one to access the value of the property of an object, when the property name is contained in a variable.
- The following notations are equivalent:

```
user.name;   // John
user['name']; // John;
var property = "name";
user[property]; // John
```

Looping in Objects (For In)

```
for (var property in user) {
  if (user.hasOwnProperty(property)) {
    console.log(property + ': ' + user[property]);
  }
}

// Output.
// name: John
// isAdmin: true
```

- The + sign is used to concatenate strings

Functions

```
// Standard function.
// Functions are reusable blocks of codes.
function showPerson(person) {
    var message = 'Hello, ';
    message = message + 'person.name';
    alert(message);
}
```

Functions

```
// Standard function.
// Functions are reusable blocks of codes.
function showPerson(person) {
    var message = 'Hello, ';
    message = message + 'person.name';
    alert(message);
}
```

This is an "input parameter."

Functions

```
// Standard function.
// Functions are reusable blocks of codes.
function showPerson(person) {
    var message = 'Hello, ';
    message = message + 'person.name';
    alert(message);
}
```

This is an "input parameter."

Note! Functions are also called "methods" or "callbacks." The definition is always the same.

Functions

```
// Execute the function.
// Remember! We have already defined
// the variable user before.
showPerson(user);
```

Functions

```
// Execute the function.
// Remember! We have already defined
// the variable user before.
showPerson(user);
```

Note! Functions are "invoked" or "executed" or "called." The terms are synonymous.

Functions

```
// Standard function.
function showPerson2(person) {
    var message = 'Hello, ';
    message = message + 'person.name';

    if (person.isAdmin === true) {
        message += 'I notice that you are an admin';
    }
    alert(message);
}
```

Functions

```
// Standard function.
function showPerson2(person) {
    var message = 'Hello, ';
    message = message + 'per

    if (person.isAdmin === true) {
        message += 'I notice that you are an admin';
    }
    alert(message);
}
```

This is an "if statement." If the condition is true, it will execute the text inside the parentheses

Functions

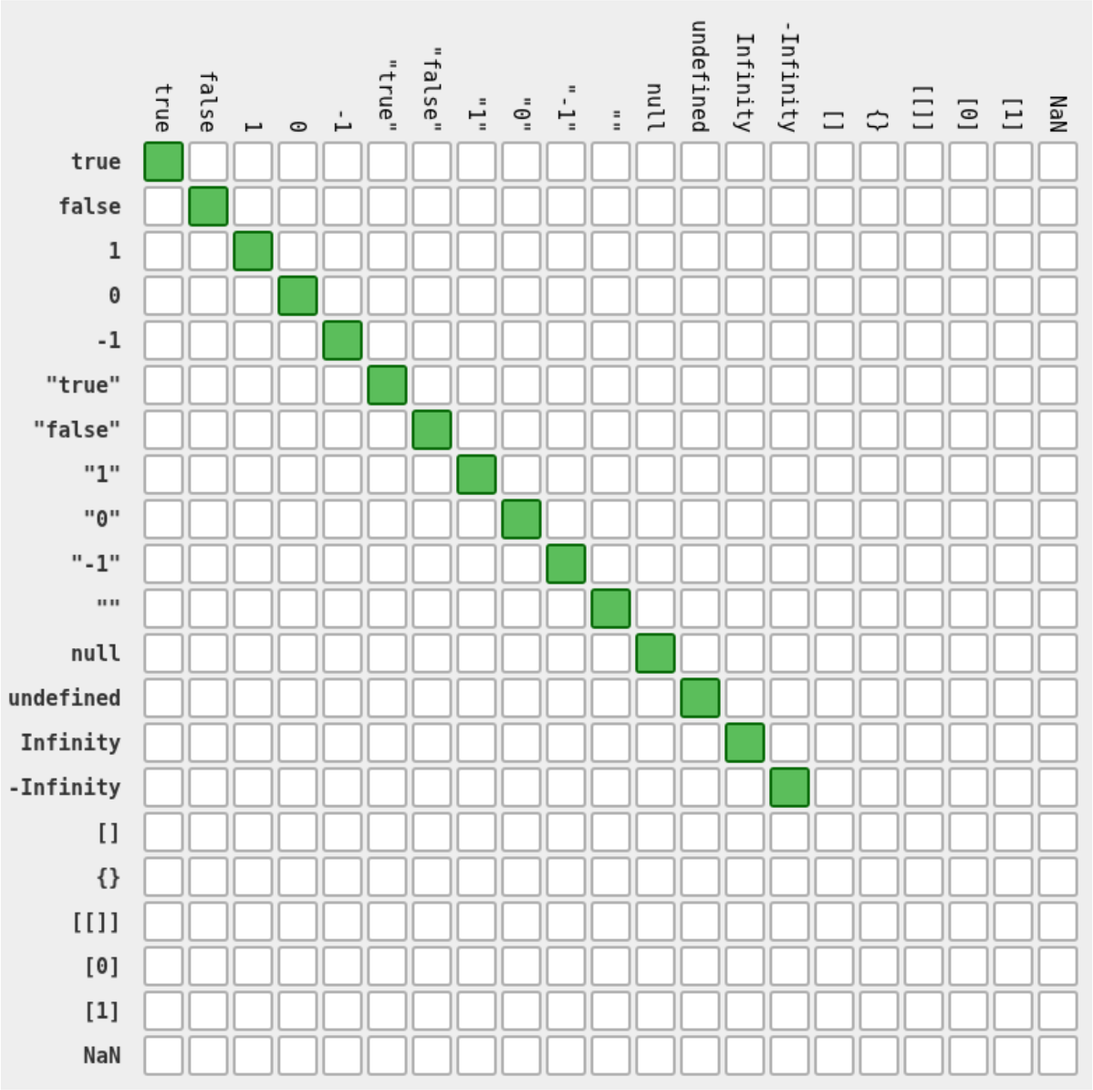
```
// Standard function.
function showPerson2(person) {
    var message = 'Hello, ';
    message = message + 'per

    if (person.isAdmin === true) {
        message += 'I notice that you are an admin';
    }
    alert(message);
}
```

The number of equals matters

- 1 equal for assignment to variables
- 2 equals for comparison
- 3 equals for **strict** comparison

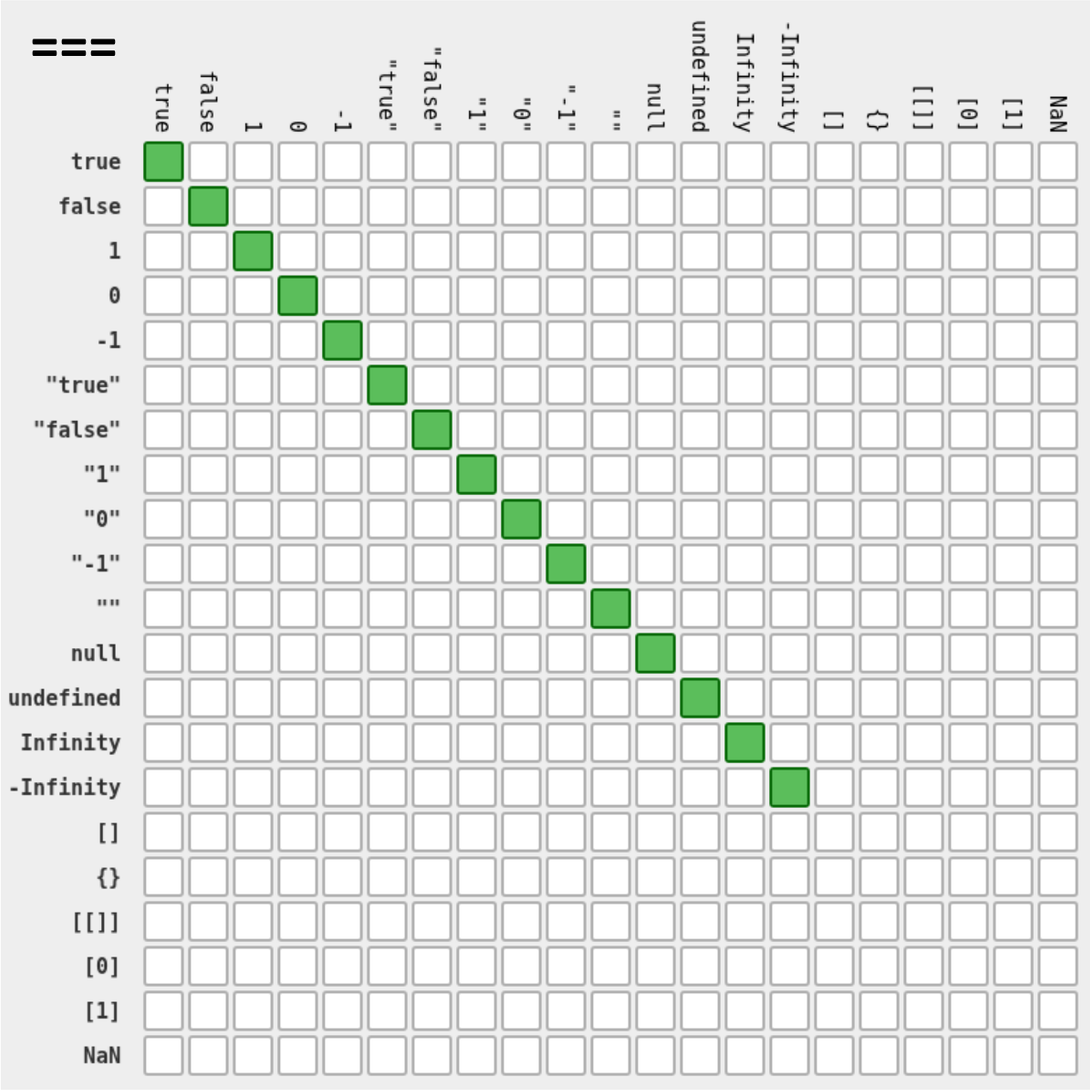
Variable Comparison: === vs ==

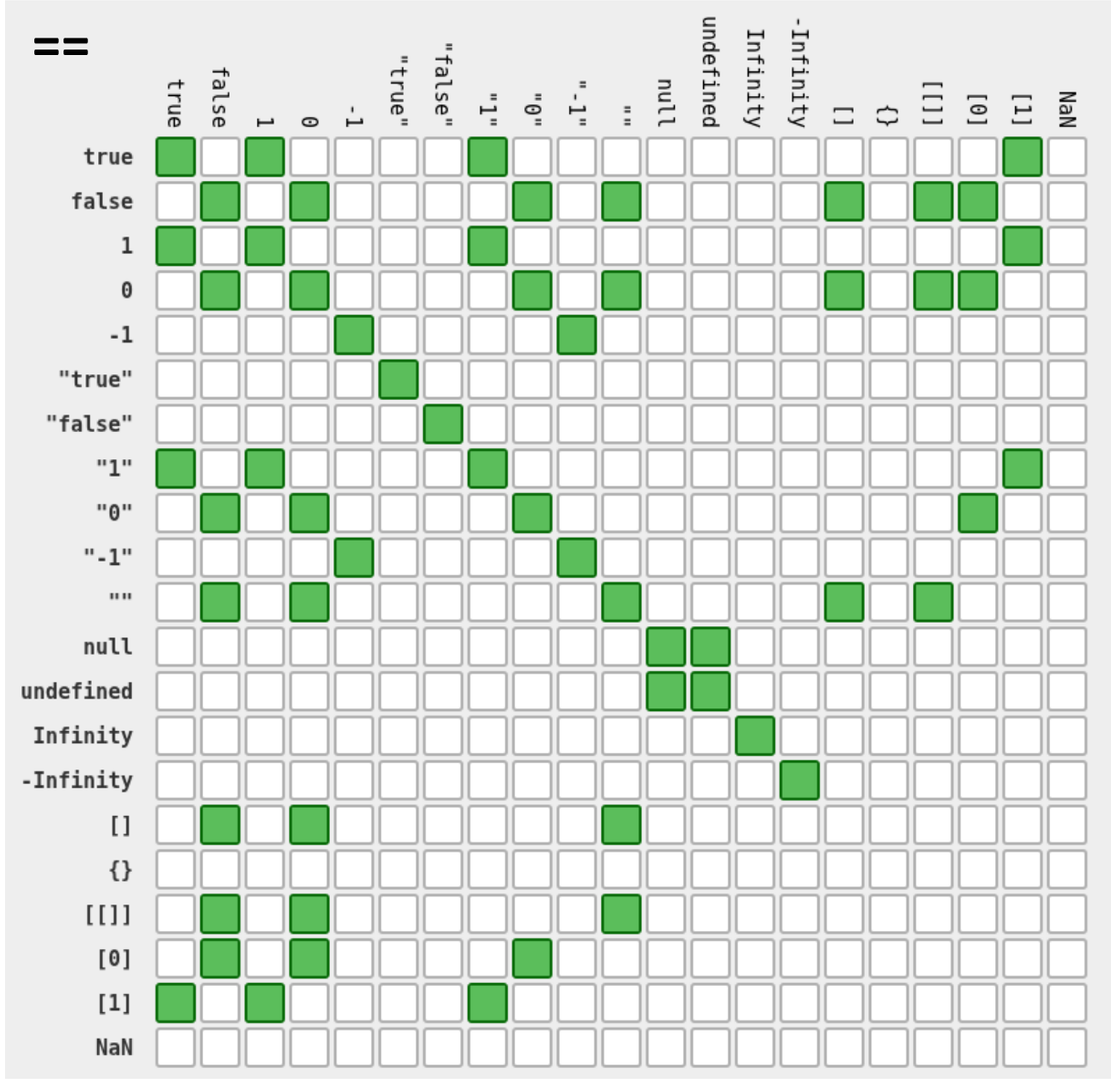
https://dorey.github.io/JavaScript-Equality-Table/



Variable Comparison: === vs ==

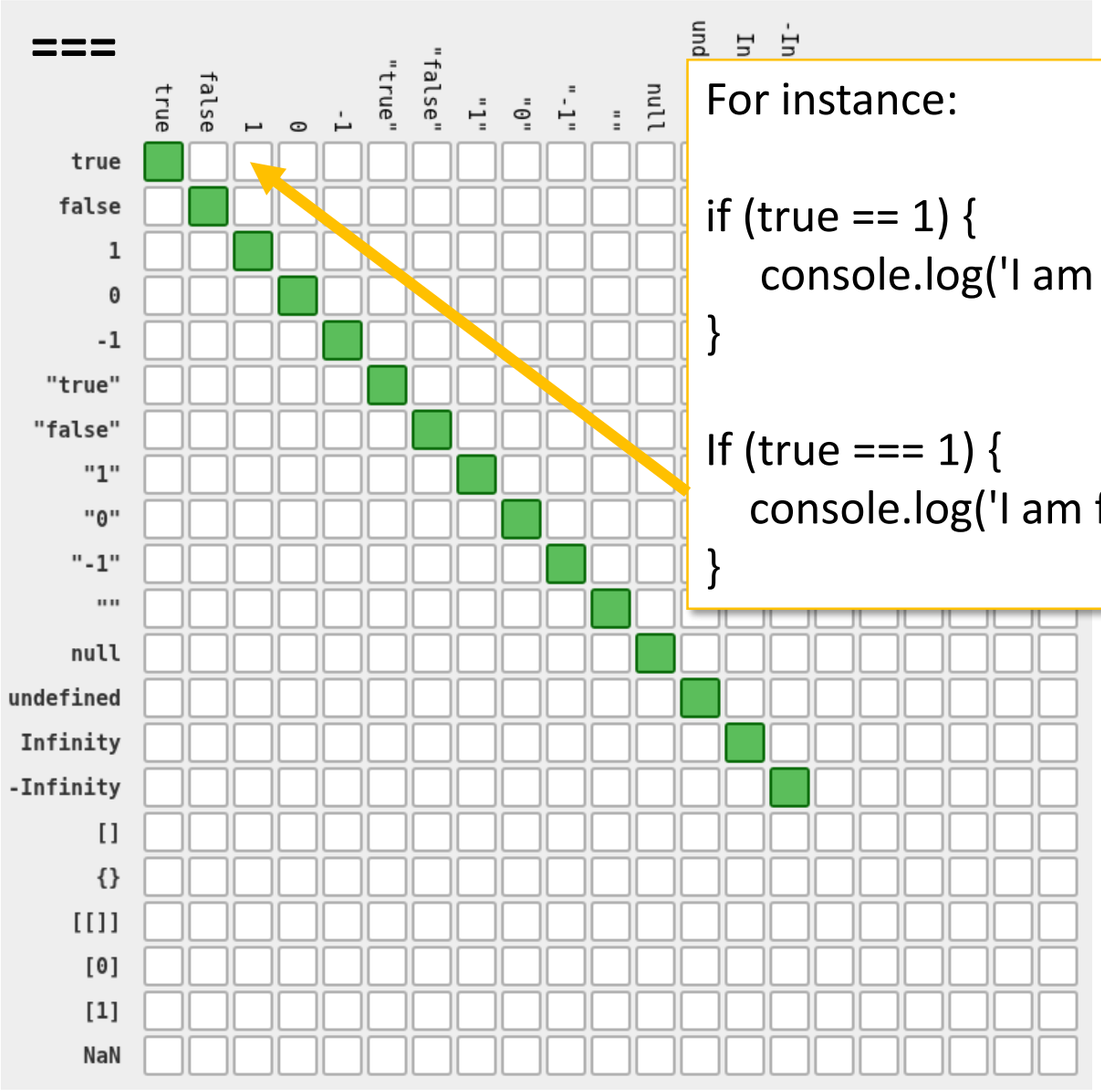
https://dorey.github.io/JavaScript-Equality-Table/





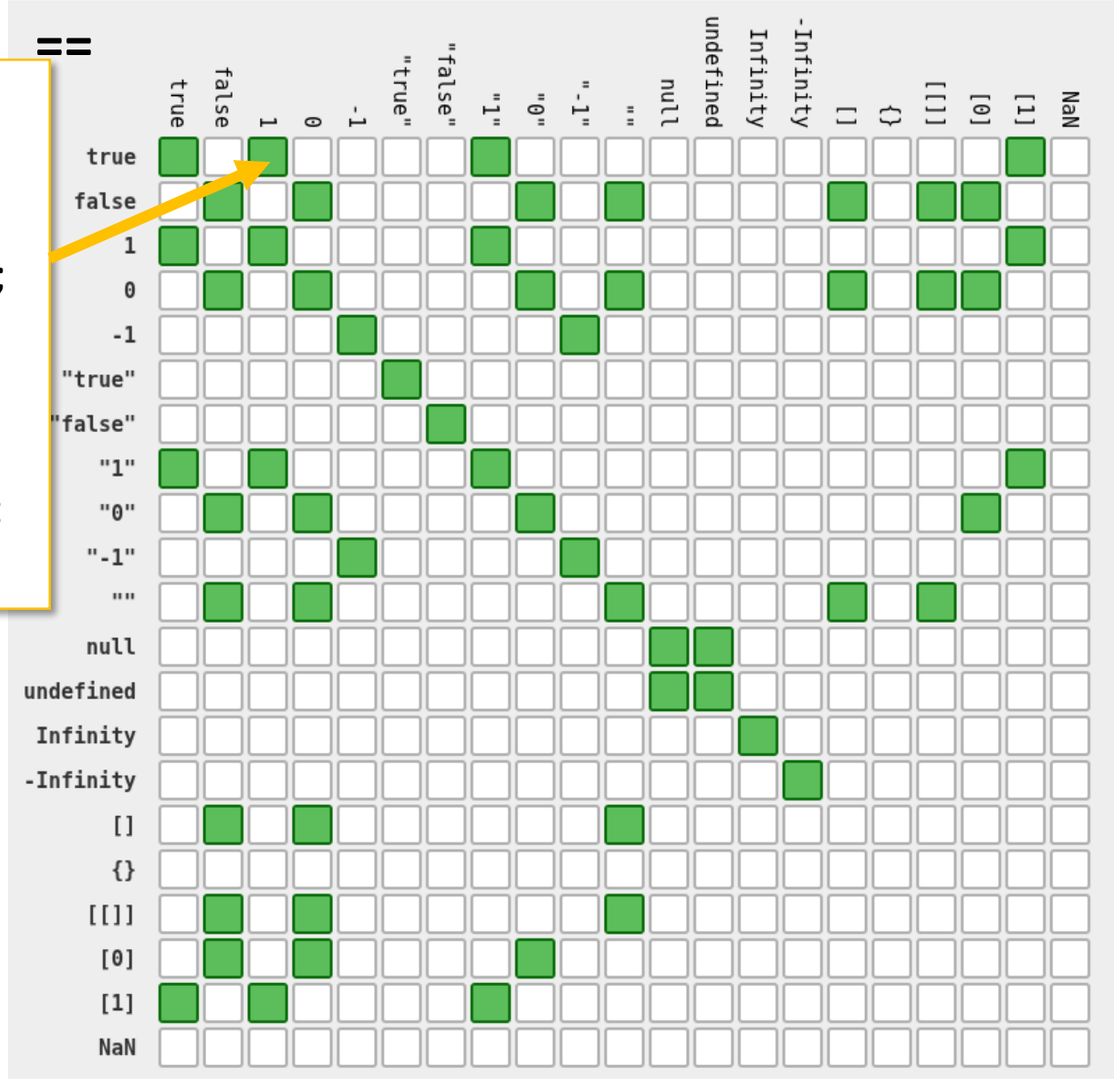
Variable Comparison: === vs ==

https://dorey.github.io/JavaScript-Equality-Table/



For instance:

```
if (true == 1) {
    console.log('I am true');
}

If (true === 1) {
    console.log('I am false');
}
```



Variable Comparison: === vs ==

https://dorey.github.io/JavaScript-Equality-Table/

==

true	■	if (true) { /* executes */ }
false		if (false) { /* does not execute */ }
1	■	if (1) { /* executes */ }
0		if (0) { /* does not execute */ }
-1	■	if (-1) { /* executes */ }
"true"	■	if ("true") { /* executes */ }
"false"	■	if ("false") { /* executes */ }
"1"	■	if ("1") { /* executes */ }
"0"	■	if ("0") { /* executes */ }
"-1"	■	if ("-1") { /* executes */ }
""		if ("") { /* does not execute */ }
null		if (null) { /* does not execute */ }
undefined		if (undefined) { /* does not execute */ }
Infinity	■	if (Infinity) { /* executes */ }
-Infinity	■	if (-Infinity) { /* executes */ }
[]	■	if ([]) { /* executes */ }
{}	■	if ({}) { /* executes */ }
[[]]	■	if ([[]]) { /* executes */ }
[0]	■	if ([0]) { /* executes */ }
[1]	■	if ([1]) { /* executes */ }
NaN		if (NaN) { /* does not execute */ }

Use always ===
(unless you have a good reason)

Arrays

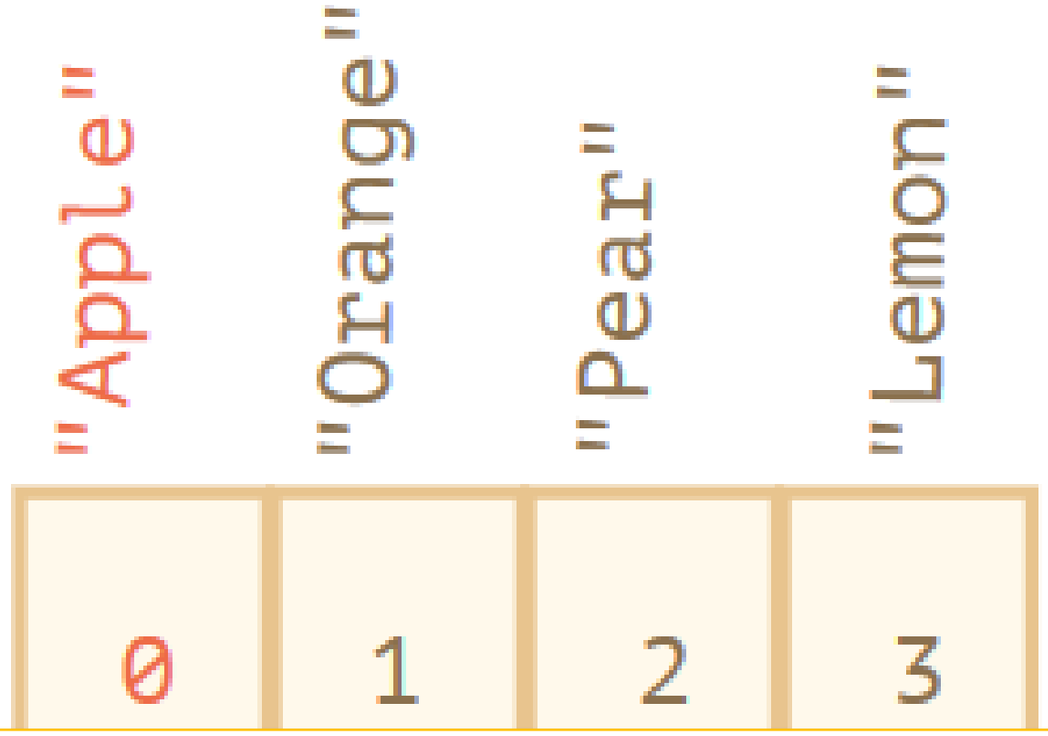
http://javascript.info/

```
var fruits = [
"Apple",
"Orange",
"Pear",
"Lemon"
];
```

Arrays

http://javascript.info/

```
var fruits = [
"Apple",
"Orange",
"Pear",
"Lemon"
];
```



Arrays are collections of items indexed by a number.
The first item has index 0, the second item has index 1, and so on…
Arrays can contain items of any type (string, number, etc.) and also mix them.

Arrays

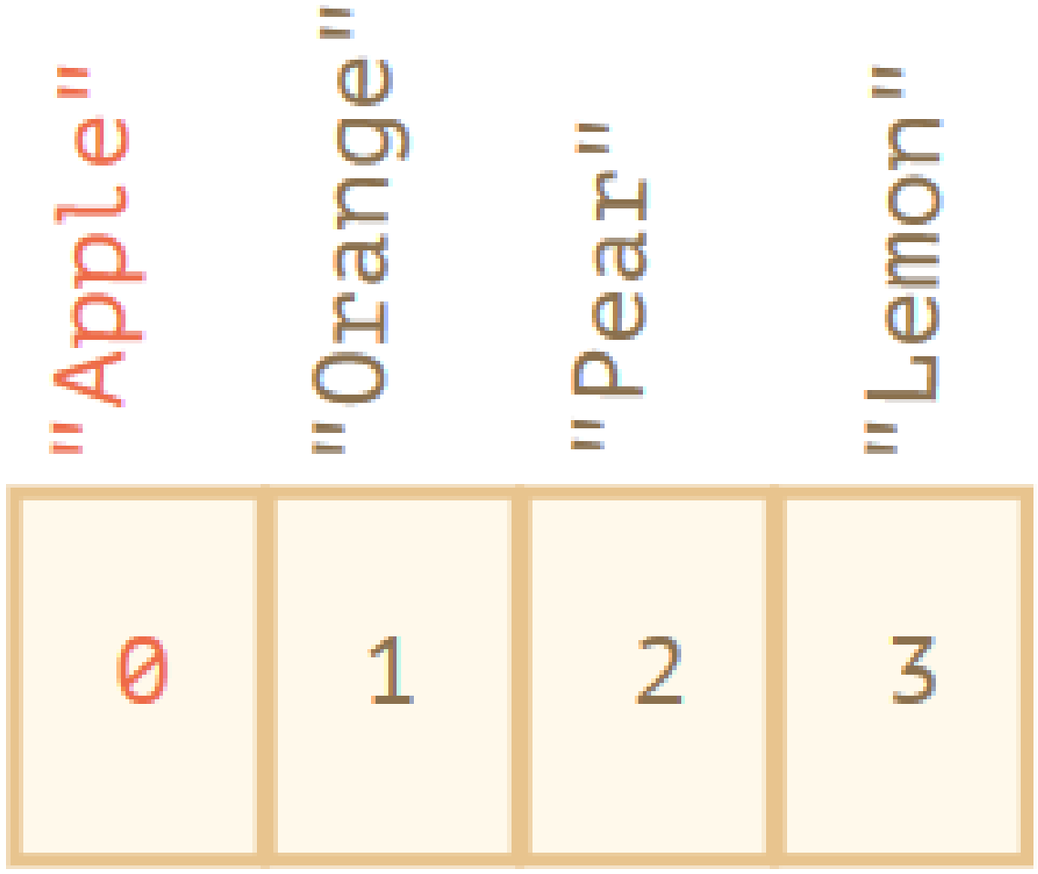
http://javascript.info/

```
var fruits = [
"Apple",
"Orange",
"Pear",
"Lemon"
];

fruits.length;
```

Arrays

http://javascript.info/

```
var fruits = [
"Apple",
"Orange",
"Pear",
"Lemon"
];

fruits.length; // 4
```



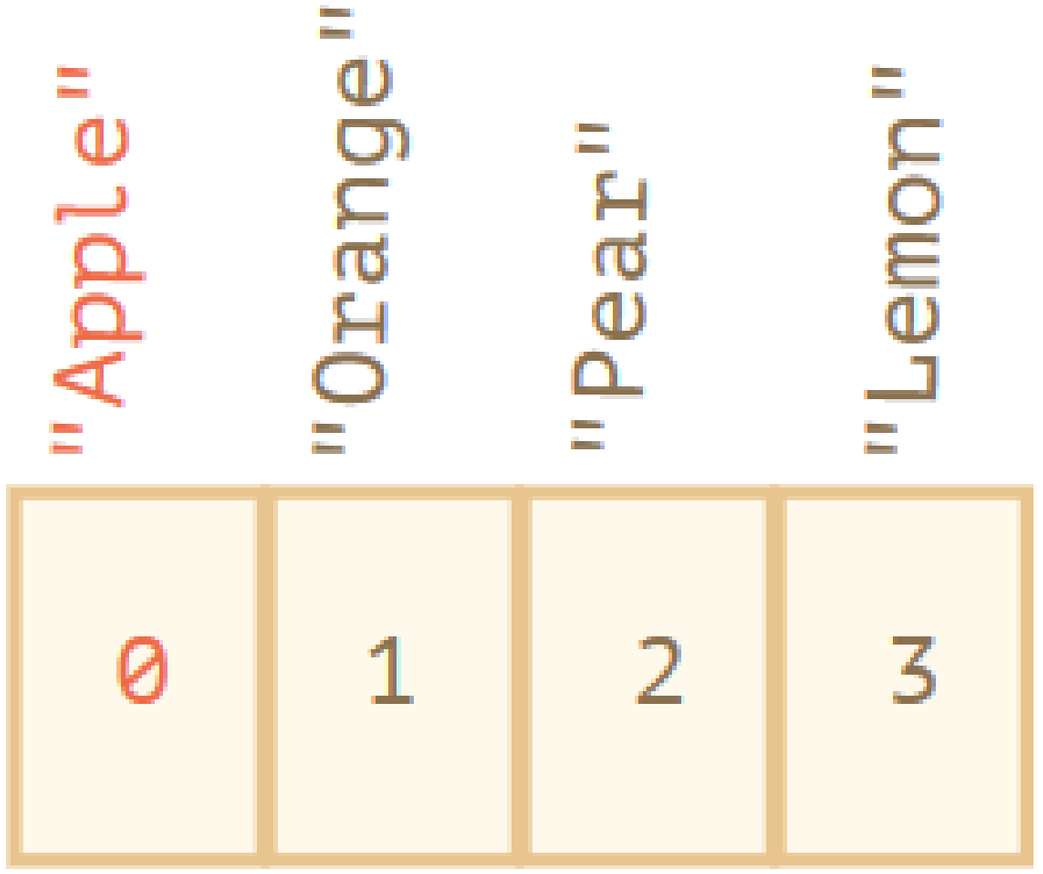
Arrays

http://javascript.info/

```
var fruits = [
"Apple",
"Orange",
"Pear",
"Lemon"
];
```



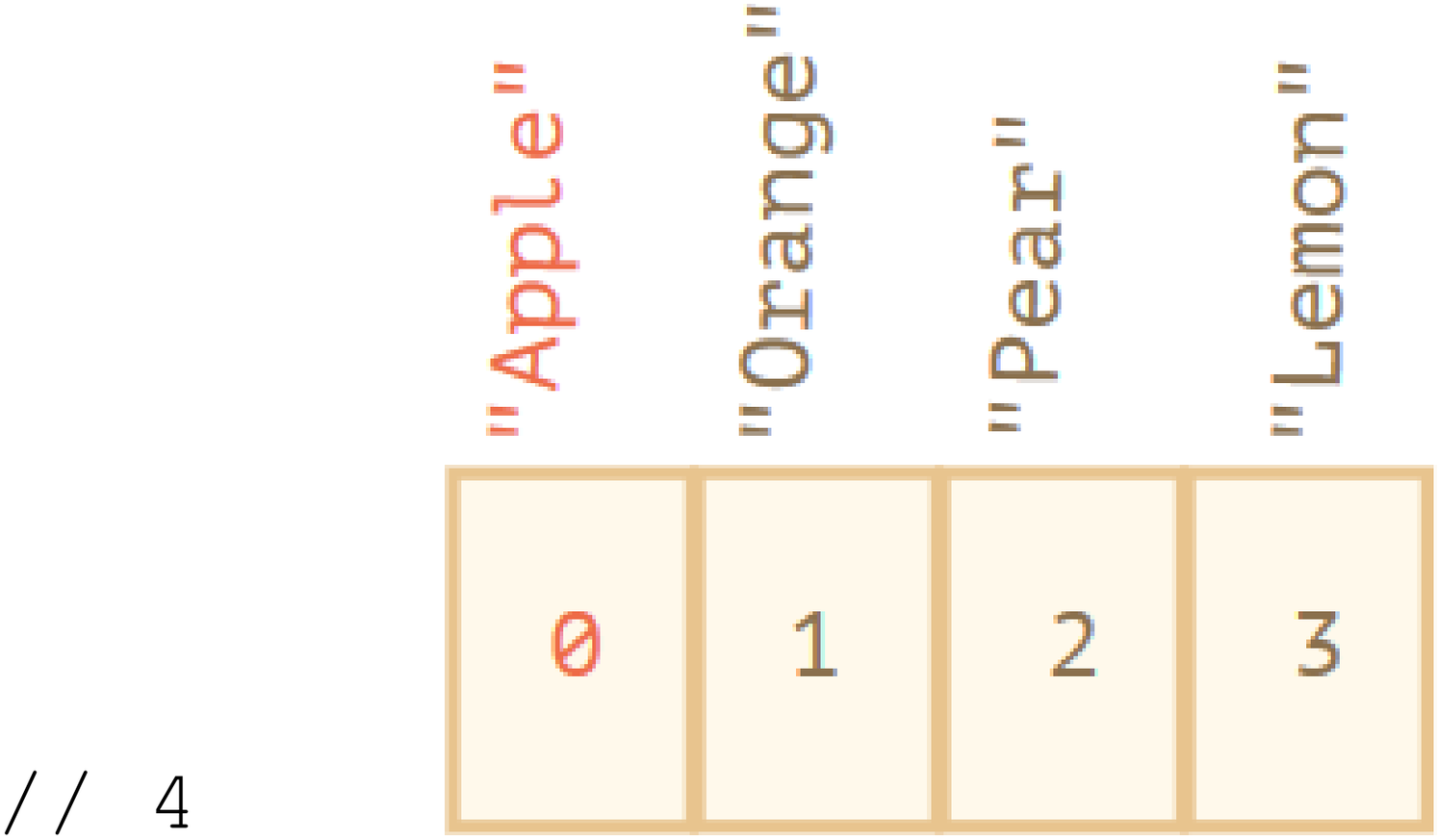
```
fruits.length; // 4
fruits[2];
```



Arrays

http://javascript.info/

```
var fruits = [
"Apple",
"Orange",
"Pear",
"Lemon"
];
```




```
fruits.length; // 4
fruits[2]; // "Pear"
```

Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];


var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {
   // Code to be added here.
}
```

Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];
```

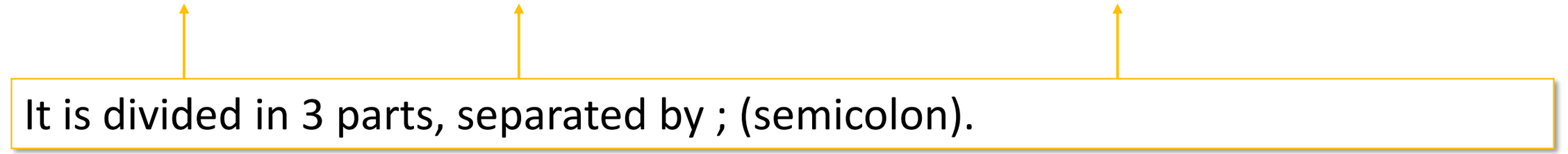
```
var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {
   // Code to be added here.
}
```

A for loop repeats the code inside the parenthesis as long as a condition is true (we will add the code later).

Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];
```

```
var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {

}
```



Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];
```

```
var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {

}
```

It is divided in 3 parts, separated by ; (semicolon).

Initialization

Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];
```

```
var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {

}
```

It is divided in 3 parts, separated by ; (semicolon).
Initialization ; **Condition**

Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];
```

```
var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {

}
```

It is divided in 3 parts, separated by ; (semicolon).
Initialization ; Condition ; **Increment (i++ means i = i + 1)**

Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];


var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {
    message += fruits[i] + ',';
}
alert(message);
```

Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];


var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {
    message += fruits[i] + ',';
}
aler
```

The first iteration i = 0, the second iteration i = 1, the third iteration i = 2, and the fourth and last iteration i = 3. In this way, we can access all the items in the array and create a text with all the fruits we like.

Arrays and For Loops

```
var fruits = [ "Apple", "Orange",
               "Pear", "Lemon" ];


var message  = 'I like ';
// This is a "for loop".
for (var i = 0 ; i < fruits.length ; i++) {
    message += fruits[i] + ',';
}
alert(mess
```

However, there is a grammatical problem! The text ends with a comma, instead that with a dot. **Do you know how to fix it?**

Main JS Operators Cheatsheet

	English Name	Usage	Example
'	Single quote	Wraps strings	`'hello'`
"	Double quote	Wraps strings	`"hello again"`
/	Slash	Comments (two in a row)	// comment
;	Semicolon	Ends a line (not mandatory, but recommended)	`'hello';`
:	Colon	Separates a key and a value in an object	{ key : 1 }
.	Dot	Access an object property (or creates it if not found)	object.key // 1
,	Comma	Separate properties in objects	{ key1 : 1 , key2 : 2 }
()	Parentheses or Brackets	Invoke a function, wrap condition statements	alert('hello'); If (counter > 10) …
[]	Square Parentheses (or Brackets)	Define an array, access elements of the array	[1, 2, 3]; array[0]; // 1
{}	Curly Parentheses (or Brackets)	Define objects, function bodies, blocks of code	{ key : 1 } function() { … } for (…) { … }