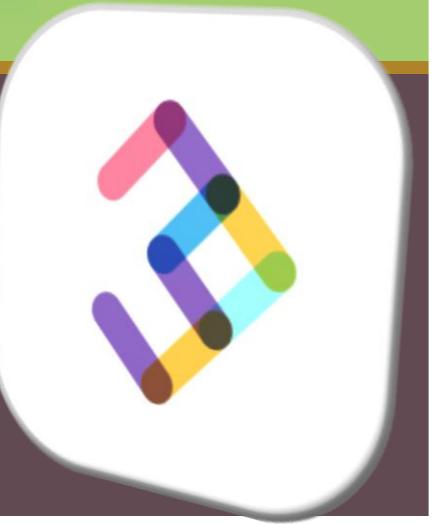




Design and Implementation of Online Behavioral Experiments



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



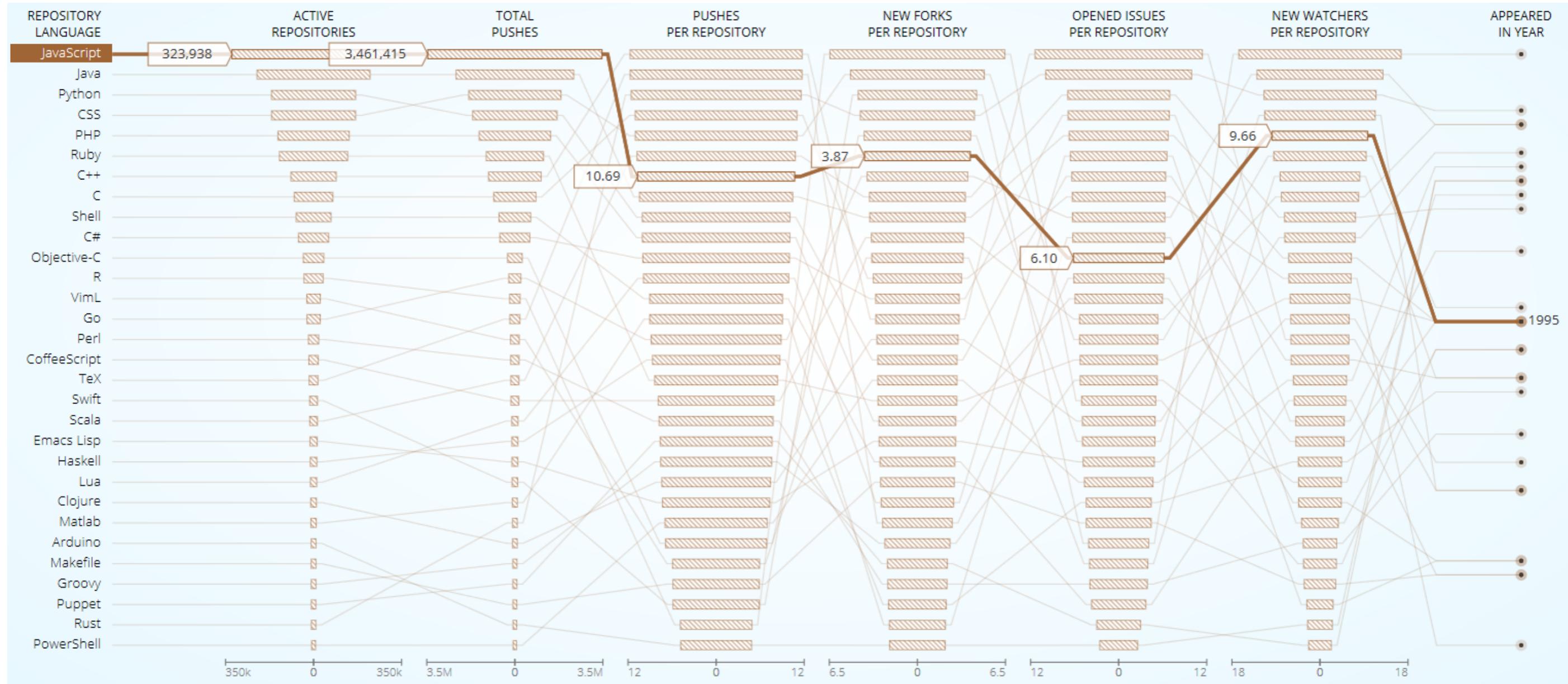
JavaScript

- 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 ECMASCIPT 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://gitut.info>

Hands On 1



Open the JavaScript console of your browser:
(ctrl+shift+l or Right Click/Inspect Element)

The screenshot shows the Chrome DevTools interface with the "Elements" tab selected. The left pane displays the DOM tree:

```
<html>
  <head></head>
  + <body>
</html>
```

The "body" element is highlighted with an orange selection bar. A tooltip message "This element has no style rules. You can" is visible above the element's style information.

The right pane shows the computed styles for the "body" element:

```
element.style {
  font-family: Ubuntu, Arial, sans-serif;
  font-size: 75%;
}

body {
  color: black;
  height: 100%;
  overflow: auto;
}
```

The "Computed Style" section also lists the "element.style" rule. The "Matched CSS Rules" section lists the "body" rule from "new_tab_theme.css:18".

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)
    ⓘ [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-ET6GsjDvpkdIkds[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

.d888b. 888 888
d88P Y88b 888 888
Y88b. 888 888 This is a browser feature intended for
"Y888b. 88888 .d88b. 88888b. 888 developers. If someone told you to copy
"Y88b. 888 d88""88b 888 "88b 888 and paste something here to enable a
"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

<!--

```
0000000          000          0000000  
111111111    11111111100    000          111111111  
00000          1111111111111111    00000          000000  
000          1111111111111111111111111111100000    000  
000          11111    1111111111111111111111100  
000          11      0    11111111100    000  
000          1      00    1    000  
000          00      00    1    000  
000          000      00000    1    000  
00000          0000      0000000    1    00000  
11111          000 00    000000    000          11111  
00000          0000      00000    00000          00000  
000          10000      00000    000          00000  
000          00000      00000    1    000  
000          000000      10000    1      0    000  
000          1000000 00    1      00    000  
000          1111111    1 0000    000  
000          1111111100    000000    000  
000          11111111111111110000000    00000  
111111111    111111111111111100000    111111111  
0000000          0000000    0000000
```

nytimes.com: a job offer!

NYTimes.com: All the code that's fit to printf()
We're hiring: <https://nytimes.wd5.myworkdayjobs.com/Tech>

-->

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()`

A screenshot of the developer tools interface showing the "Console" tab selected. A large orange circle highlights the "Clear" button, which is represented by a trash can icon. Below the tabs, there are several filter buttons: Errors, Warnings, Logs, Info, and Debug. To the right, there are buttons for CSS, XHR, Requests, and Persist Logs. At the bottom left, there is a "»" button followed by a vertical line.

Hands On 1



Screenshot of a browser developer tools interface showing the 'Console' tab selected. The console area contains the command 'console.log('This is my very own text');'. Below the console, the output shows the text 'This is my very own text'.

Inspector Console Debugger Network Style Editor Performance Memory Storage > ... X

Filter output Errors Warnings Logs Info Debug CSS XHR Requests Persist Logs

» |

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.

A screenshot of a browser's developer tools interface. The tabs at the top are: Inspector (highlighted with an orange circle), Console, Debugger, Network, Style Editor, Performance, Memory, Storage, and more. Below the tabs are buttons for Filter output, Errors, Warnings, Logs, Info, Debug, CSS, XHR, Requests, and Persist Logs. A search bar with a double arrow icon is at the bottom left.

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.

A screenshot of a browser's developer tools toolbar. The tabs shown are Inspector (highlighted with an orange circle), Console, Debugger, Network, Style Editor, Performance, Memory, Storage, and more. Below the toolbar are buttons for Filter output, Errors, Warnings, Logs, Info, Debug, and categories for CSS, XHR, Requests, and Persist Logs. A yellow arrow points from the explanatory text below to the 'Inspector' tab.

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>

Images and Links

<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.

≡ POLITICS

The New York Times

SUBSCRIBE NOW LOG IN

Trump Impeachment Inquiry Formal Vote Planned on Impeachment Inquiry Witness Ignored Congressional Subpoena Why Political Influence in Foreign Policy Matters W

Shifting Course, Democrats Plan First Floor Vote on Impeachment Inquiry

Keep reading The Times by creating a free account or logging in.

Google Facebook

OR

Hands On 1: Browser's Dev Tools



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

A screenshot of the browser developer tools interface. The top navigation bar includes tabs for Inspector, Console, Debugger, Network, Style Editor, Performance, Memory, Storage, and more. The 'Console' tab is highlighted with an orange circle. Below the tabs are buttons for Filter output, Errors, Warnings, Logs, Info, Debug, CSS, XHR, Requests, and Persist Logs. A text input field at the bottom left contains two double arrows (») followed by a vertical bar |.

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.

The screenshot shows the browser's developer tools open, specifically the element inspector. The DOM tree is visible, with the body element selected. A yellow arrow points from the text "the selected element will be hidden in the page." in the box on the left to the body element in the DOM tree. Another yellow arrow points from the text "the selected element will be hidden in the page." in the box on the left to the "display: none;" line in the CSS rules panel. The CSS rules panel shows the following styles:

```
element { display: none; }  
body { overflow-x: hidden; }
```

The browser's address bar shows the URL: <https://static01.nyt.com/ads/tpc-check.html>.

JavaScript



Great tutorial from novice to JavaScript Ninja:

<http://javascript.info/>

Variables

Variables

<http://javascript.info/>

Variables

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



Variables

<http://javascript.info/>

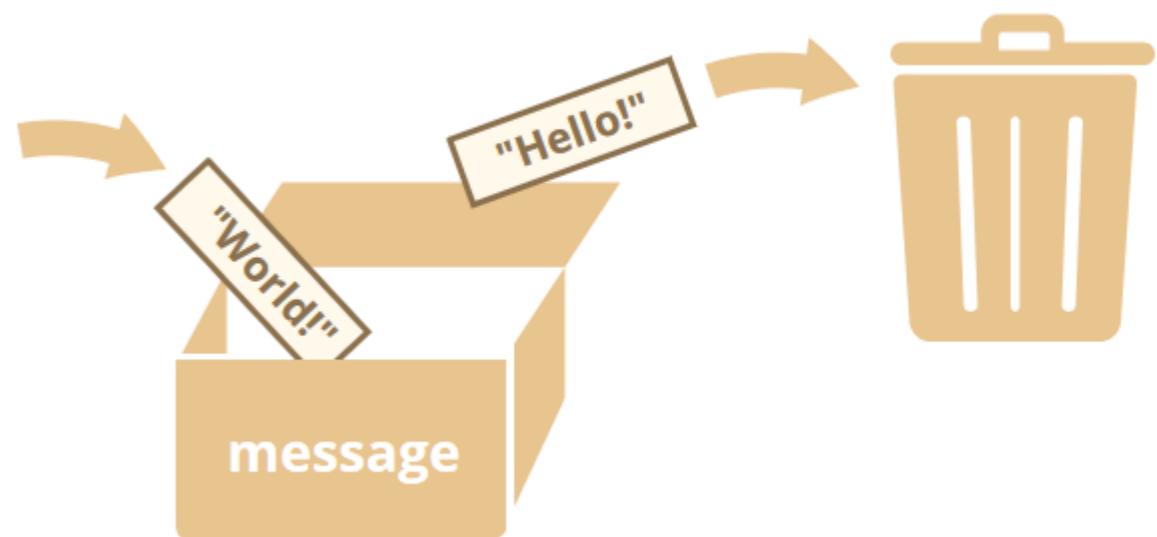
Variables

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



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

```
alert(message);
```



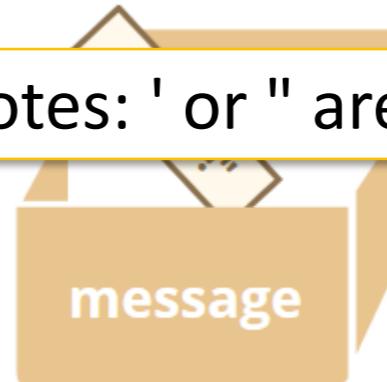
Variables

<http://javascript.info/>

Variables

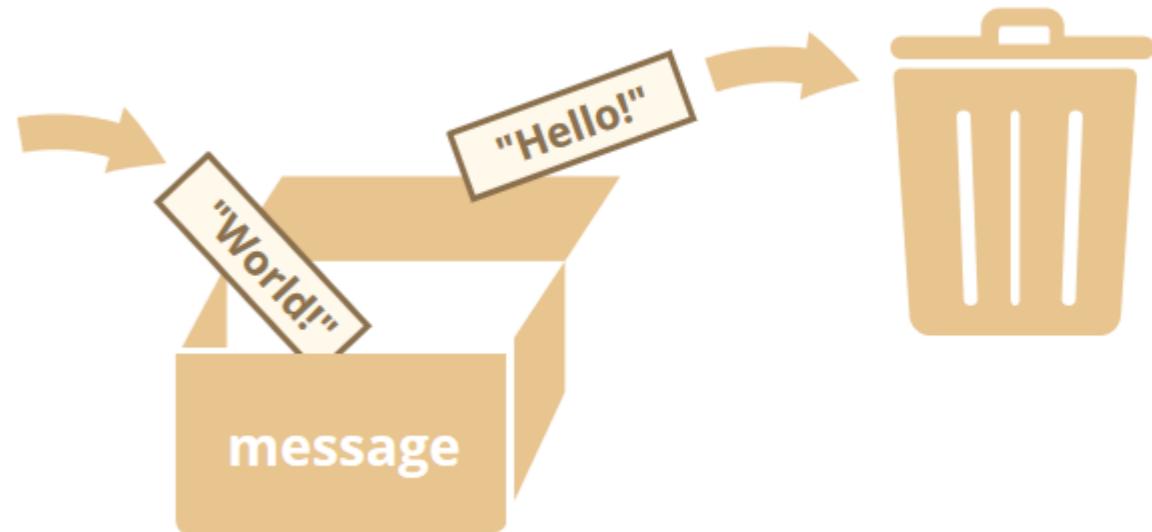
Strings must be wrapped in quotes: ' or " are equivalent.

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



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

```
alert(message);
```



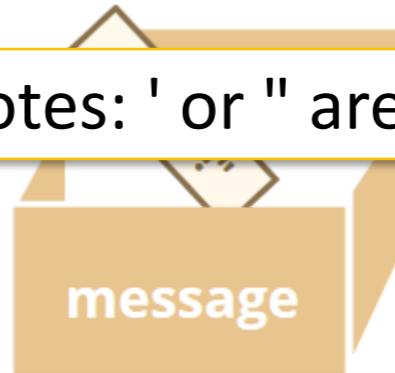
Variables

<http://javascript.info/>

Variables

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

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

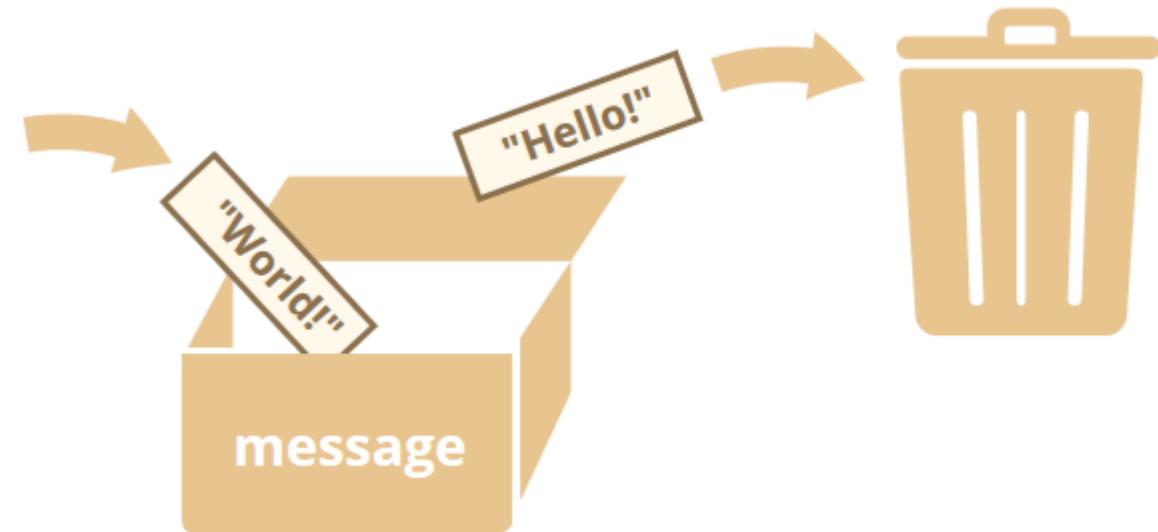


Text following // is a comment and it is not read by JavaScript

```
// value changed.
```

```
message = 'World!';
```

```
alert(message);
```



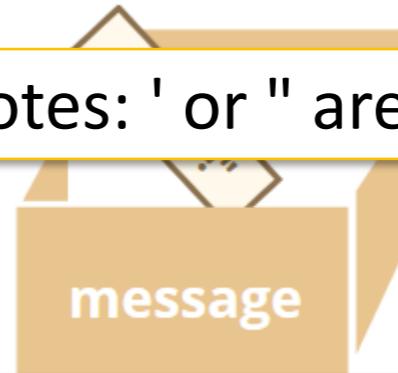
Variables

<http://javascript.info/>

Variables

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

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



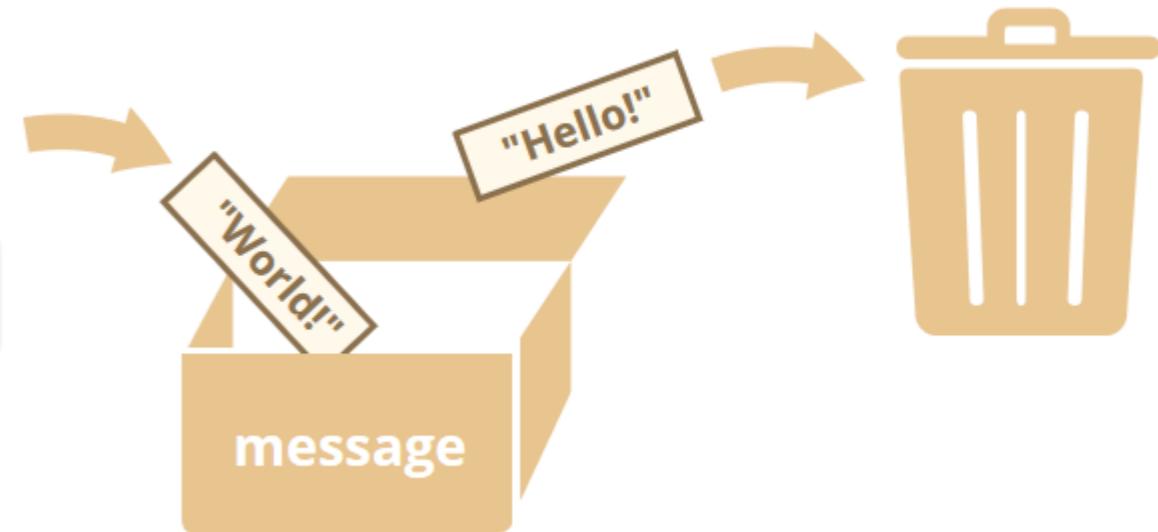
Text following // is a comment and it is not read by JavaScript

```
// 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);
```

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

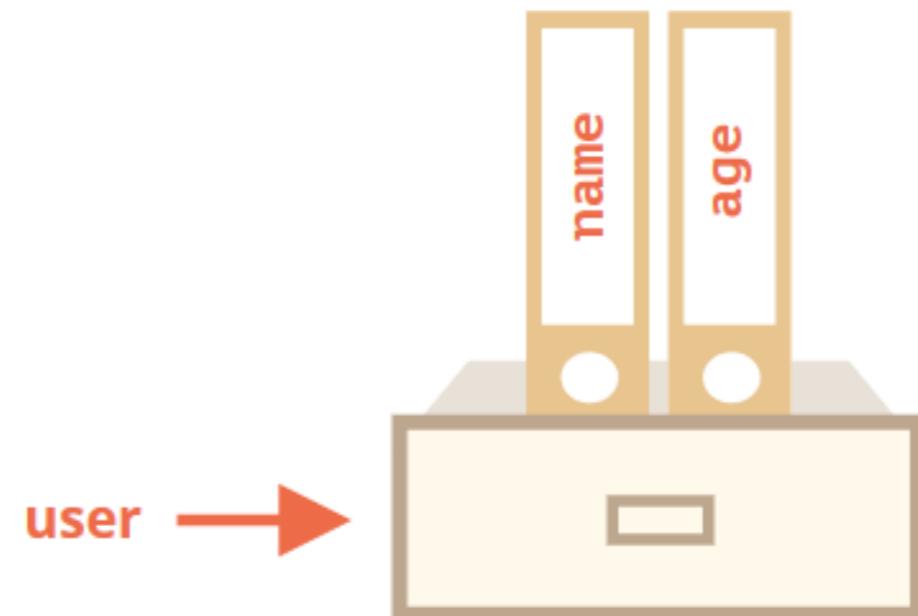
```
// type of value changed to number  
message = 2019;
```

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

Objects

<http://javascript.info/>

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

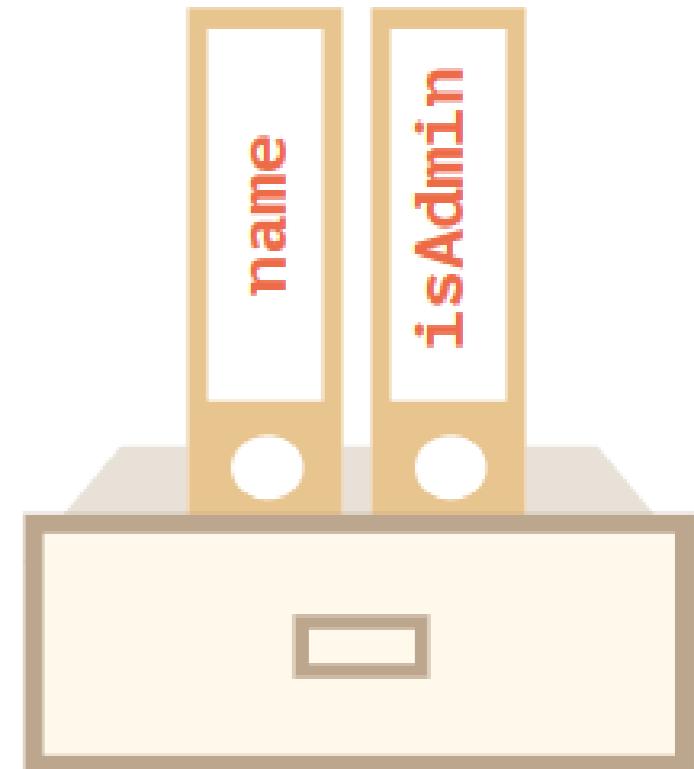


Objects

<http://javascript.info/>

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

user →



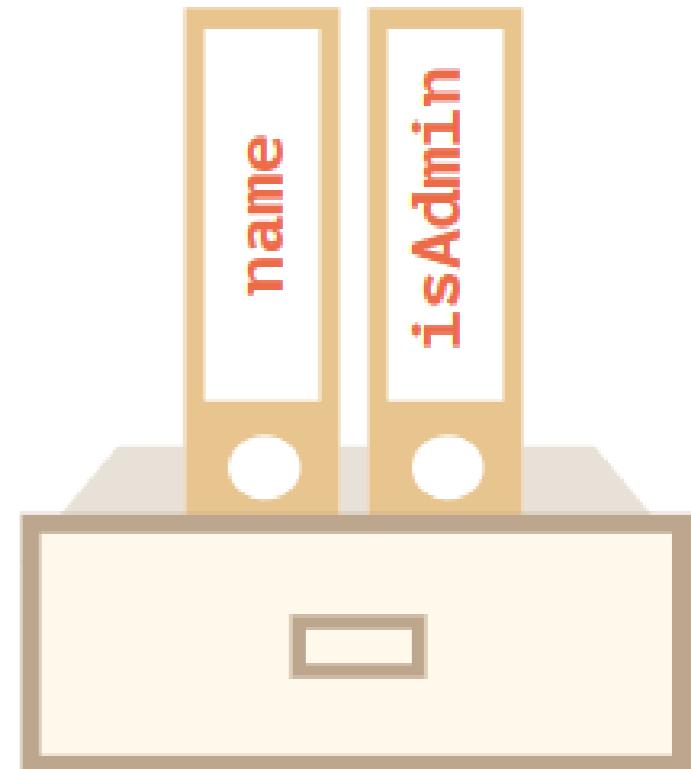
Objects

<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.

user →



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]);  
  }  
}
```

- The + sign is used to concatenate strings

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

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 ==

The figure is a heatmap illustrating the results of the JavaScript equality operator (`==`) across a grid of 20 values on both the horizontal and vertical axes. The values are:

- Primitives:
 - NaN
 - [1]
 - [0]
 - []
 - {}
 - Infinity
 - Infinity
 - undefined
 - null
 - ""
 - "-1"
 - "0"
 - "1"
 - "true"
 - "false"
 - "1"
 - "0"
 - "-1"
 - ..."
- Objects:
 - true
 - false
 - 1
 - 0
 - 1
 - 1
 - 0
 - 1
 - true
 - false
 - 1
 - 0
 - 1
 - ...
 - null
 - undefined
 - Infinity
 - Infinity
 - []
 - {}
 - [[]]
 - [0]
 - [1]
 - NaN

A green square at the intersection of row `a` and column `b` indicates that `a == b`. Most comparisons result in false, shown as white squares.

Variable Comparison: === vs ==

The figure is a heatmap illustrating the results of comparing various JavaScript values using the equality operator (`==`). The x-axis and y-axis both list the following items:

- Primitives: `true`, `false`, `1`, `0`, `-1`, `"true"`, `"false"`, `"1"`, `"0"`, `"-1"`, `""`, `null`, `undefined`, `Infinity`, `-Infinity`.
- Objects: `[]`, `{}`, `[[]]`, `[0]`, `[1]`, `NaN`.

A green square at the intersection of `true` on both axes indicates that `true == true` is true. Most other comparisons result in false, indicated by white squares.

Variable Comparison: === vs ==

==		!=		<		>		<=		>=	
		-In	In	und							
true	true	true	false	true	true	false	false	true	false	true	false
false	false	false	true	false	false	true	true	false	true	false	true
1	1	1	0	1	1	0	0	1	0	1	0
0	0	0	1	0	0	1	1	0	1	0	1
-1	-1	-1	0	-1	-1	0	0	-1	0	-1	0
"true"	"true"	"true"	"false"	"true"	"true"	"false"	"false"	"true"	"true"	"true"	"false"
"false"	"false"	"false"	"true"	"false"	"false"	"true"	"true"	"false"	"false"	"false"	"true"
"1"	"1"	"1"	"0"	"1"	"1"	"0"	"0"	"1"	"1"	"1"	"0"
"0"	"0"	"0"	"1"	"0"	"0"	"1"	"1"	"0"	"0"	"0"	"1"
"-1"	"-1"	"-1"	"0"	"-1"	"-1"	"0"	"0"	"-1"	"-1"	"-1"	"0"
""	""	""	"1"	""	""	"0"	"0"	""	""	""	"1"
null	null	null	1	null	null	0	0	null	null	null	1
undefined	undefined	undefined	0	undefined	undefined	1	1	undefined	undefined	undefined	0
Infinity	Infinity	Infinity	1	Infinity	Infinity	0	0	1	1	1	0
-Infinity	-Infinity	-Infinity	0	-Infinity	-Infinity	1	1	0	0	0	1
[]	[]	[]	1	[]	[]	0	0	1	1	1	0
{}	{}	{}	0	{}	{}	1	1	0	0	0	1
[[]]	[[]]	[[]]	1	[[]]	[[]]	0	0	1	1	1	0
[0]	[0]	[0]	0	[0]	[0]	1	1	0	0	0	1
[1]	[1]	[1]	1	[1]	[1]	0	0	1	1	1	0
NaN	NaN	NaN	1	NaN	NaN	0	0	1	1	1	0

For instance:

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

==		!=		<		>		<=		>=	
		-In	In	und							
true	true	true	false	true	true	false	false	true	false	true	false
false	false	false	true	false	false	true	true	false	true	false	true
1	1	1	0	1	1	0	0	1	0	1	0
0	0	0	1	0	0	1	1	0	1	0	1
-1	-1	-1	0	-1	-1	0	0	-1	0	-1	0
"true"	"true"	"true"	"false"	"true"	"true"	"false"	"false"	"true"	"true"	"true"	"false"
"false"	"false"	"false"	"true"	"false"	"false"	"true"	"true"	"false"	"false"	"false"	"true"
"1"	"1"	"1"	"0"	"1"	"1"	"0"	"0"	"1"	"1"	"1"	"0"
"0"	"0"	"0"	"1"	"0"	"0"	"1"	"1"	"0"	"0"	"0"	"1"
"-1"	"-1"	"-1"	"0"	"-1"	"-1"	"0"	"0"	"-1"	"-1"	"-1"	"0"
""	""	""	"1"	""	""	"0"	"0"	""	""	""	"1"
null	null	null	1	null	null	0	0	null	null	null	1
undefined	undefined	undefined	0	undefined	undefined	1	1	undefined	undefined	undefined	0
Infinity	Infinity	Infinity	1	Infinity	Infinity	0	0	1	1	1	0
-Infinity	-Infinity	-Infinity	0	-Infinity	-Infinity	1	1	0	0	0	1
[]	[]	[]	1	[]	[]	0	0	1	1	1	0
{}	{}	{}	0	{}	{}	1	1	0	0	0	1
[[]]	[[]]	[[]]	1	[[]]	[[]]	0	0	1	1	1	0
[0]	[0]	[0]	0	[0]	[0]	1	1	0	0	0	1
[1]	[1]	[1]	1	[1]	[1]	0	0	1	1	1	0
NaN	NaN	NaN	1	NaN	NaN	0	0	1	1	1	0

Variable Comparison: === vs ==

==	
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/>

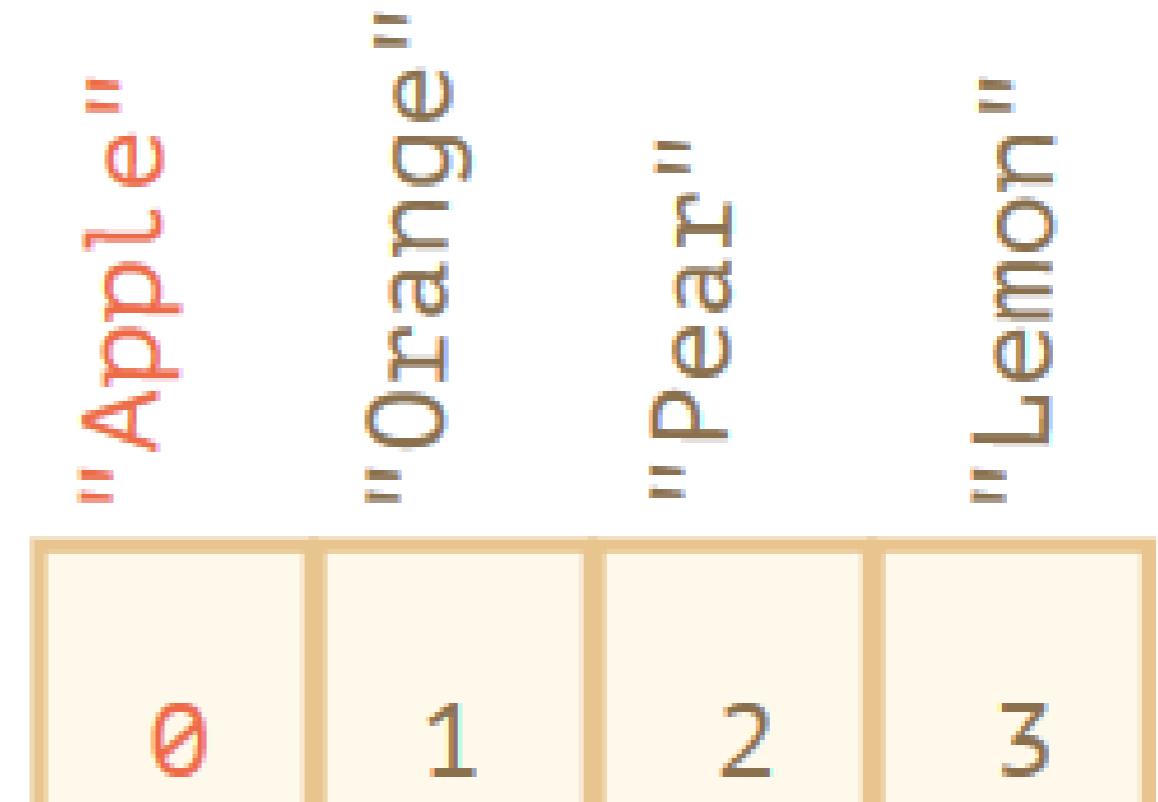
```
var fruits = [  
  "Apple",  
  "Orange",  
  "Pear",  
  "Lemon"  
];
```



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++) {  
}
```

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

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] + ',';
}
```

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(message);
```

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 (...) { ... }