CORSO DI LAUREA IN INFORMATICA

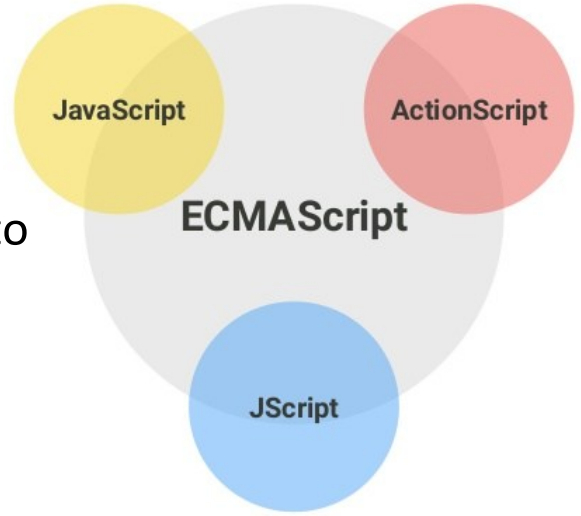
Tecnologie Software per il Web

JAVASCRIPT

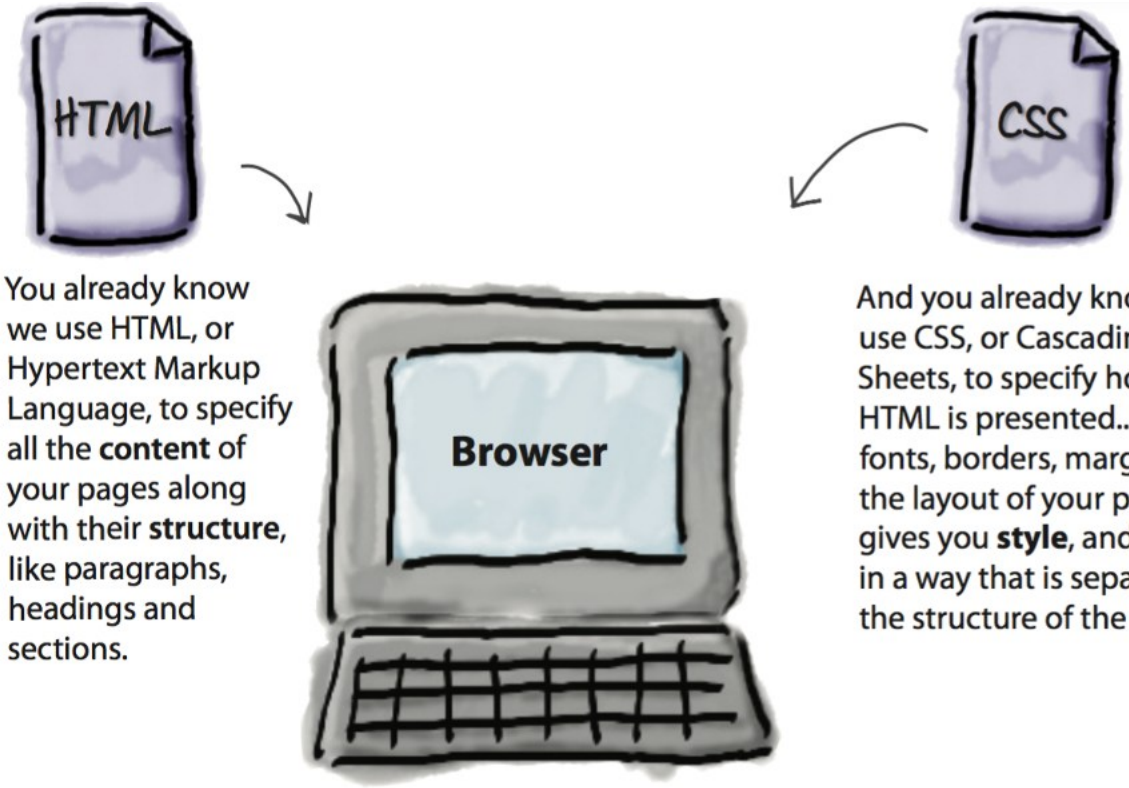
Docente: prof. Romano Simone

a.a. 2024-2025

Che cos'è JavaScript



- JavaScript è un linguaggio di **scripting** sviluppato per dare interattività alle pagine HTML
- Può essere inserito direttamente nelle pagine Web ed è in pratica lo standard client-side
- Il suo nome ufficiale è ECMAScript
  - È diventato standard ECMA (European Computer Manufactures Association) (ECMA-262) nel 1997
  - È anche uno standard ISO (ISO/IEC 16262)
- Sviluppato inizialmente da Netscape (il nome originale era LiveScript) e introdotto in Netscape 2 nel 1995
- In seguito anche Microsoft ha lavorato sul linguaggio producendo una sua variante chiamata JScript



You already know we use HTML, or Hypertext Markup Language, to specify all the **content** of your pages along with their **structure**, like paragraphs, headings and sections.

And you already know that we use CSS, or Cascading Style Sheets, to specify how the HTML is presented...the colors, fonts, borders, margins, and the layout of your page. CSS gives you **style**, and it does it in a way that is separate from the structure of the page.

So let's introduce JavaScript, HTML & CSS's computational cousin. JavaScript lets you create **behavior** in your web pages. Need to react when a user clicks on your "On Sale for the next 30 seconds!" button? Double check your user's form input on the fly? Grab some tweets from Twitter and display them? Or how about play a game? Look to JavaScript. JavaScript gives you a way to add programming to your page so that you can compute, react, draw, communicate, alert, alter, update, change, and we could go on... anything dynamic, that's JavaScript in action.

Processo di standardizzazione di JavaScript

- È diventato standard ECMA nel 1997 (ECMA-262)
- Nel dicembre 1999 si è giunti alla versione ECMA-262 Edition 3, anche noto come ECMAScript Edition 3, corrisponde a JavaScript 1.5
- Nel dicembre 2009 si è definita la versione ECMAScript Edition 5 (superset di ECMAScript Edition 3), corrispondente a JavaScript 1.8
- Nel giugno 2011 si è giunti ECMAScript Edition 5.1 (superset di ECMAScript Edition 5), corrispondente a JavaScript 1.8.5
- ECMAScript 6 (2015)
- ECMAScript 7 (2016)
- ECMAScript 8 (2017)
- ECMAScript 9 (2018)
- ECMAScript 10 (2019)
- ECMAScript 11 (2020)
- ECMAScript 12 (2021)
- ECMAScript 13 (2022)
- ECMAScript 14 (2023)
- ECMAScript 15 (2024)

JavaScript e Java

- *Java e JavaScript sono due cose completamente diverse*
- L'unica similitudine è legata al fatto di aver entrambi adottato la sintassi del C
- Esistono profonde differenze
  - JavaScript è **interpretato** e non compilato
  - JavaScript è principalmente **object-based** (solo dall'ECMAScript 6 è diventato anche **class-based**)
    - Si possono creare oggetti senza definire classi
  - JavaScript è **debolmente tipizzato** *(weakly typed)*
    - I tipi sono assegnati ai dati implicitamente

JavaScript testing

- Problem
  - **Java**: very strict compliance tests to be called “java”
    - You can have very high confidence that code written in Java 8/9 on Windows version will run identically (except for some differences in how GUIs look) on Java 8/9 on Macos, Linux, Solaris, and other windows versions. True for Java from Oracle, Apple, IBM, or open-source versions
  - **JavaScript**: every browser vendor makes own version, with no outside checks
    - Behavior of same JavaScript program can vary substantially from browser to browser, and even from one release of the same browser to another
- Consequence
  - Before final deployment, you must test on all browsers you expect to support
  - Most developers
    - Do initial testing and development on either ***Chrome*** or ***Firefox***
    - But test also on Internet Explorer, Microsoft Edge, and Safari before final deployment

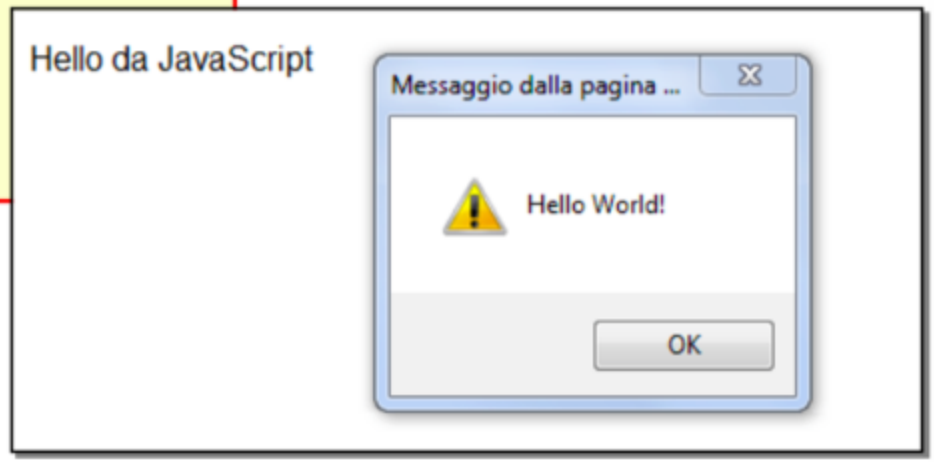
Cosa si può fare con JavaScript

- Il codice JavaScript viene eseguito da un interprete contenuto all'interno del browser
- Nasce per dare **dinamicità** alle pagine Web
- Consente quindi di:
  - Accedere e modificare elementi della pagina HTML
  - Reagire ad eventi generati dall'interazione fra utente e pagina
  - Validare i dati inseriti dall'utente
  - Interagire con il browser: determinare il browser utilizzato e la dimensione della finestra in cui viene mostrata la pagina, lavorare con i browser cookie, ecc.

Esempio

- Vediamo la versione JavaScript dell'ormai mitico *HelloWorld!*
- Viene mostrato un popup con la scritta HelloWorld
- Lo script viene inserito nella pagina HTML usando il tag **<script>**:

```
<html>
  <body>
    <p>Hello da JavaScript</p>
    <script type="text/javascript">
       alert("Hello World!");
    </script>
  </body>
</html>
```



alert() show a message in a dialog box

Example: setTimeout and function (test.html in javascript.zip)

Here's our standard HTML5 doctype, and <html> and <head> elements.

And we've got a pretty generic <body> for this page as well.

Ah, but we've added a script element to the <head> of the page.

And we've written some JavaScript code inside it.

Again, don't worry too much about what this code does. Then again, we bet you'll want to take a look at the code and see if you can think through what each part might do.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Just a Generic Page</title>
    <script>
      setTimeout(wakeUpUser, 5000);
      function wakeUpUser() {
         alert("Are you going to stare at this boring page forever?");
      }
    </script>
  </head>
  <body>
       <h1>Just a generic heading</h1>
       <p>Not a lot to read about here. I'm just an obligatory paragraph living in
an example in a JavaScript book. I'm looking for something to make my life more
exciting.</p>
  </body>
</html>
```

Sintassi del linguaggio

- La sintassi di JavaScript è modellata su quella del C con alcune varianti significative
- In particolare
  - È un linguaggio **case-sensitive**
  - Le istruzioni sono terminate da **';'** ma il terminatore può essere omesso se si va **a capo**
  - Sono ammessi sia commenti multilinea (delimitati da **/*** e ***/**) che monolinea (iniziano con **//**)
- Gli identificatori possono contenere lettere, cifre e i caratteri **'_'** e **'$'** ma non possono iniziare con una cifra
- Le parole chiave non possono essere usate per definire i nomi delle variabili

Variabili

- Le variabili possono essere dichiarate usando la parola chiave **var**, **let** o **const** ma anche senza specificare **nessuna parola chiave**

  var nomevariabile;

  let nomevariabile;

  const nomevariabile; (per le constanti!!!)

  nomevariabile;

- **Le variabili non hanno un tipo**
  - possono contenere valori di qualunque tipo
- Si può inizializzare contestualmente alla dichiarazione, esempio:

  var f = 15.8;

- Possono essere dichiarate in linea:

  for (var i = 1; i<10; i++) { … }

var, let, const o nessuna parola chiave?

- Regola generale:
  - Dichiara sempre le variabili con **const** se il valore della variabile non deve cambiare
  - Altrimenti usa **let** o **var**
    - La maggior parte dei programmatori JavaScript oggi una **let**
  - Evita di dichiarare le variabili senza specificare nessuna parola chiave
- **let** e **const** sono stati introdotti con ECMAScript 6 nel 2015
  - browser precedenti all'uscita dell'ECMAScript 6 non supportano **let** e **const**
- Nota che esistono differenze negli scope delle variabili dichiarate con **var**, **let**, **const**

Scope delle variabili

- Lo scope delle variabili determina la visibilità delle variabili
- JavaScript ha 3 tipi di scope
  - **Blocco** (variabili dichiarate all'interno di un blocco { } non sono accessibili all'esterno del blocco)
  - **Funzione** (variabili dichiarate all'interno di una funzione non sono accessibili all'esterno di essa)
  - **Globale** (si può accedere ovunque alle variabili, vivono fintantoché non si chiude la pagina)
- **Let** e **const** hanno visibilità a livello di blocco, **var** no

```
{
  let x = 2;
}
// x can NOT be used here
```

```
{
  var x = 2;
}
// x CAN be used here
```

Scope delle variabili (2)

- Le variabili dichiarate con **var**, **let** e **const** quando dichiarate all'interno di una funzione hanno visibilità a livello della funzione

```
function myFunction() {
  var carName = "Volvo";   // Function Scope
}

function myFunction() {
  let carName = "Volvo";   // Function Scope
}

function myFunction() {
  const carName = "Volvo";   // Function Scope
}
```

Scope delle variabili (3)

- Le variabili dichiarate **senza parola alcuna chiave** all'interno di una funzione hanno visibilità globale

```
myFunction();

// code here can use carName

function myFunction() {
  carName = "Volvo";
}
```

Scope delle variabili (4)

- Le variabili dichiarate con **var**, **let** e **const** quando dichiarate globalmente (ovvero al di fuori di funzioni) hanno visibilità globale (anche all'interno di funzioni)

```
var x = 2;        // Global scope

let x = 2;        // Global scope

const x = 2;       // Global scope
```

Alcuni esempi

```
var x;

x = 6;
```

```
var x = 5 + 6;
var y = x * 10;
```

```
var x = 5;
var y = 6;
var z = x + y;
```

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

```
x = x + 5
```

```
var x = 5;   // I will be executed

// var x = 6;   I will NOT be executed
```

```
var person = "John Doe", carName = "Volvo", price = 200;
```

```
var  x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

```
const PI = 3.141592653589793;
PI = 3.14;      // This will give an error
PI = PI + 10;   // This will also give an error
```

```
var  x = 10;
// Here x is 10
{
  const x = 2;
  // Here x is 2
}
// Here x is 10
```

Alcune keywords JavaScript

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

JavaScript keywords (*ECMAScript 6)

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const*	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

If: one, two, or more options

- Single option

```
if (condition) {
  ...
}
```

- Two options

```
if (condition) {
  ...
} else {
  ...
}
```

- More than two options

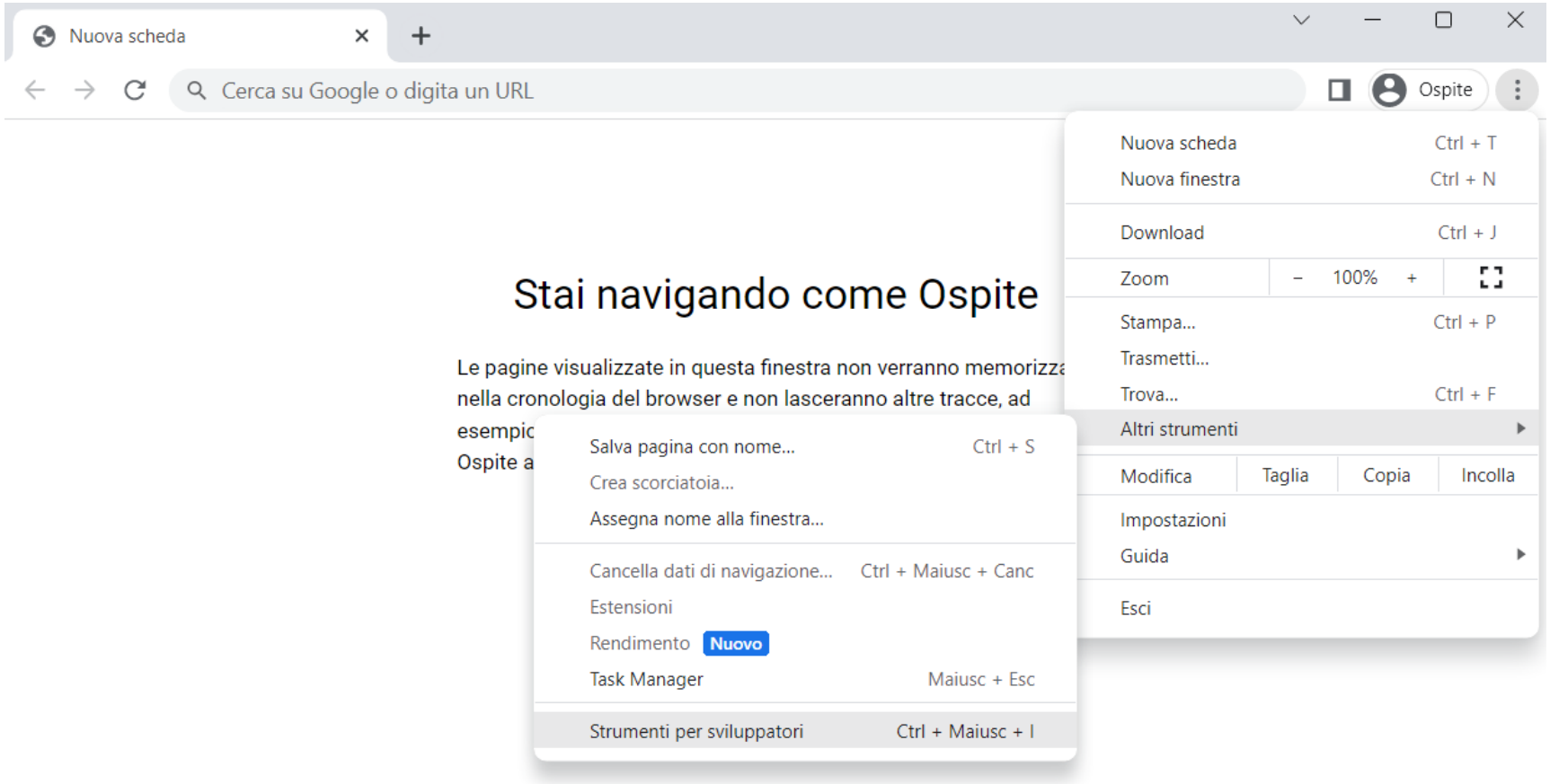
```
if (condition) {
  ...
} else if (condition) {
  ...
} else {
  ...
}
```

JavaScript has a liberal definition of what condition is “false” (fails the test):

- “false”: false, null, undefined, "" (empty string), 0, NaN
- “true”: anything else

Opening the console

- Every browser has a slightly different implementation of the console. And, to make things even more complicated, the way that browsers implement the console changes fairly frequently

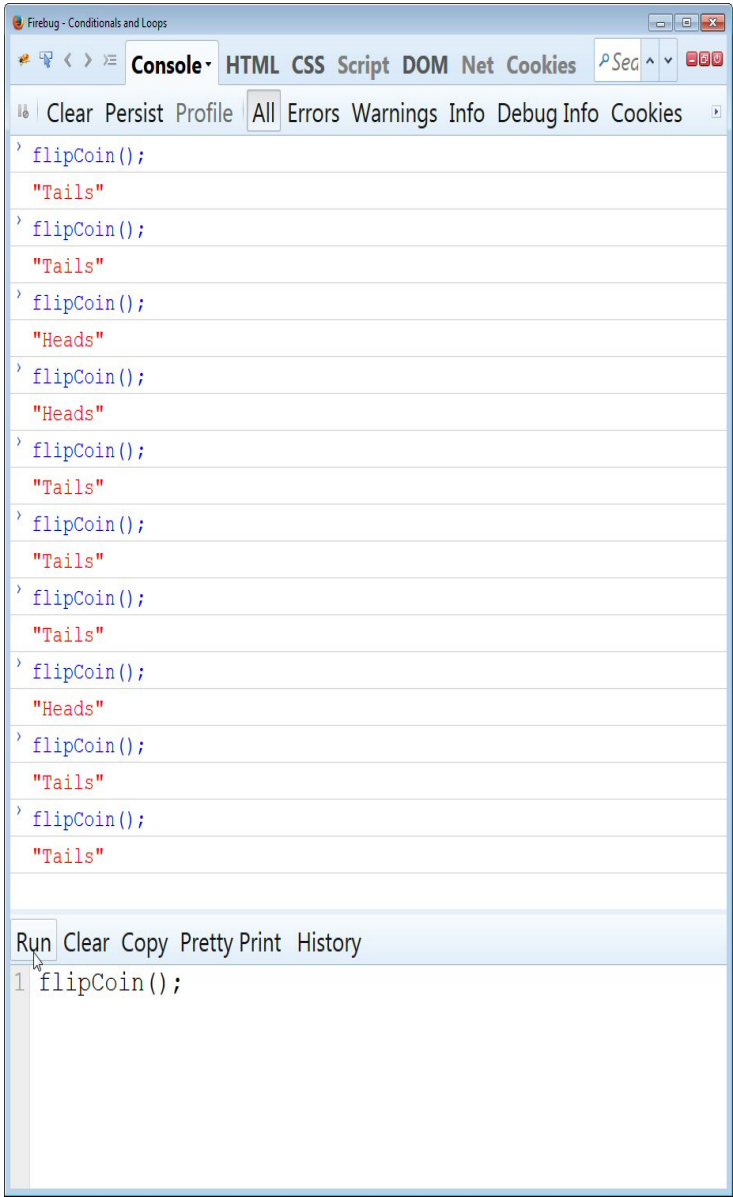


Example: if-else

```
function flipcoin() {
  if (Math.random() < 0.5) {
    return("heads");
  } else {
    return("tails");
  }
}
alert(flipcoin());
```

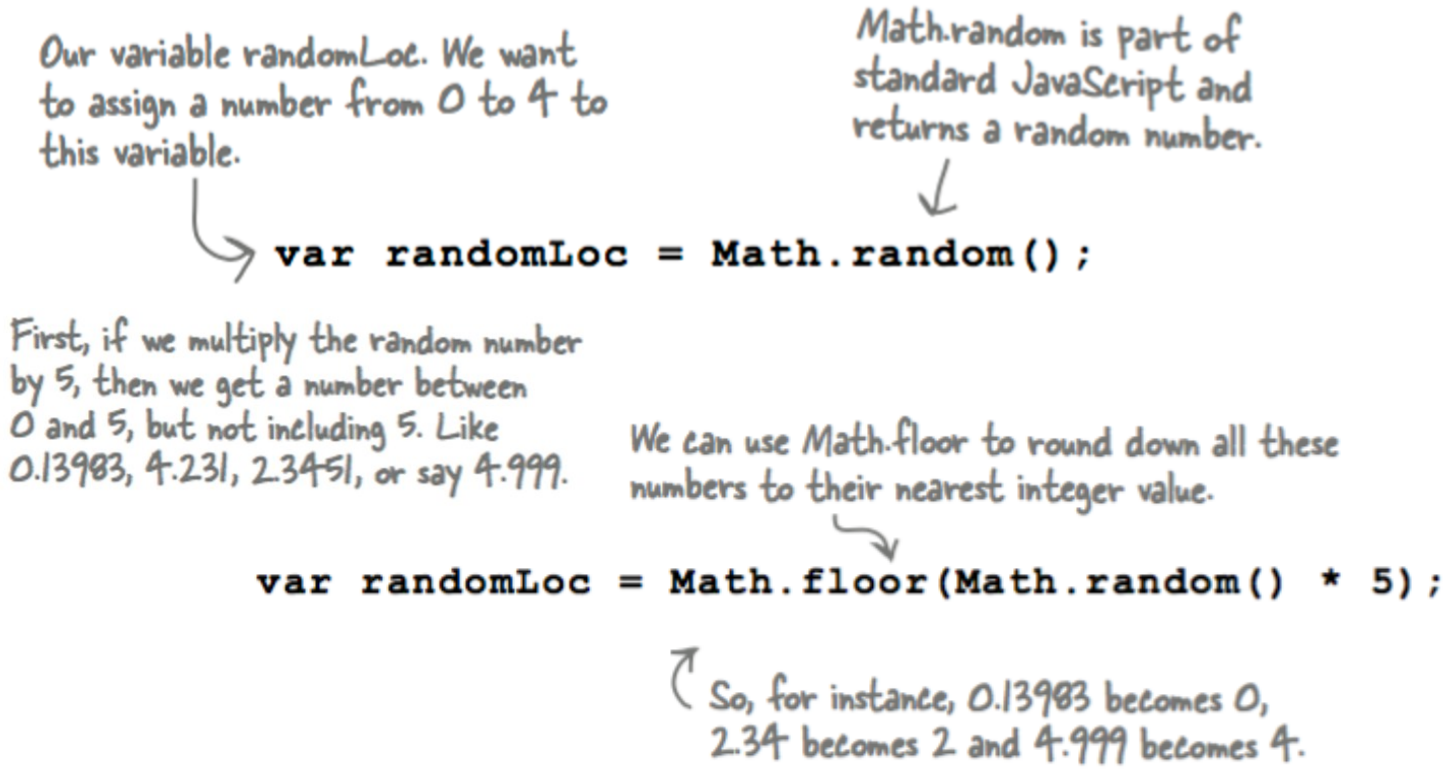
Math.random() returns a number between 0.0 (inclusive) and 1.0 (exclusive). If the random number generator is good, the values should be evenly distributed and unpredictable

Type «allow pasting» in the console to allow pasting



Random

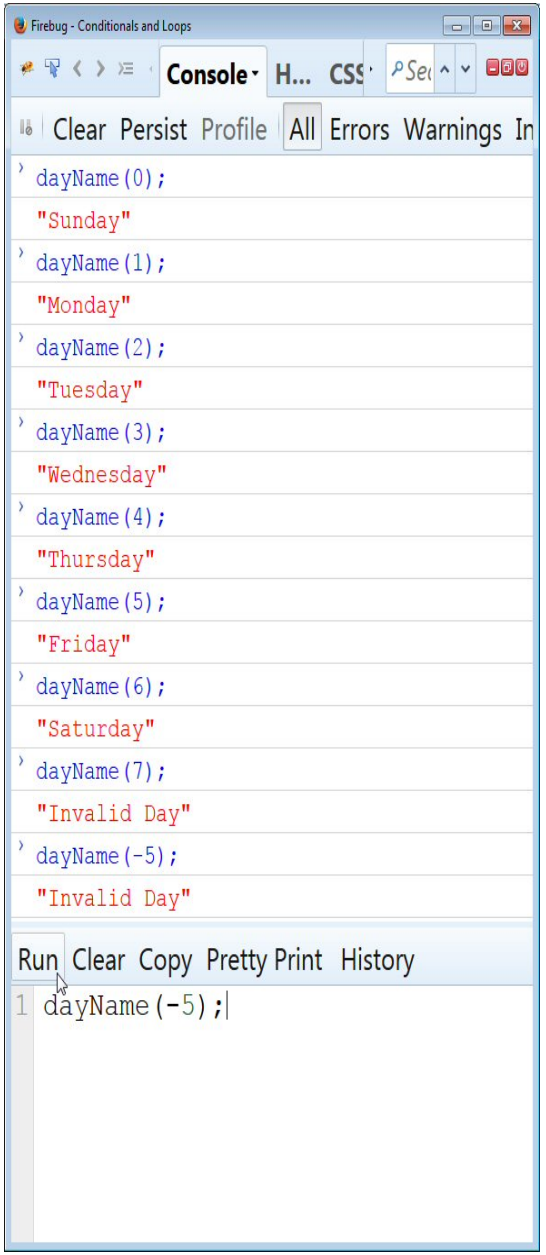
- We need is an integer between 0 and 4



In a range [min, max]: **Math.floor(Math.random() * (max - min + 1)) + min**

Switch statement

```
function dayname(daynumber) {
  var dayname;
  switch(daynumber) {
    case 0: dayname = "sunday"; break;
    case 1: dayname = "monday"; break;
    case 2: dayname = "tuesday"; break;
    case 3: dayname = "wednesday"; break;
    case 4: dayname = "thursday"; break;
    case 5: dayname = "friday"; break;
    case 6: dayname = "saturday"; break;
    default: dayname = "invalid day";
             break;
  }
  return(dayname);
}
```



JavaScript comparison and logical operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

it attempts to convert and compare operands that are of different types:
console.log('1' == 1) gives true

console.log('1' === 1) gives false

```
x = (1 < 2) ? true : false;
```

&&, ||	Logical **and**, **or**. Both use ***short-circuit evaluation***
!	Logical **negation**

JavaScript arithmetic and unary operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

```
x = 5 + 5;
y = "5" + 5;
z = "Hello" + 5;
```

```
var x = 5;
var y = 2;
var z = x % y;
```

```
(5 + 6) * 10
```

**	Exponentiation	10 ** 2	(ECMAScript 7)

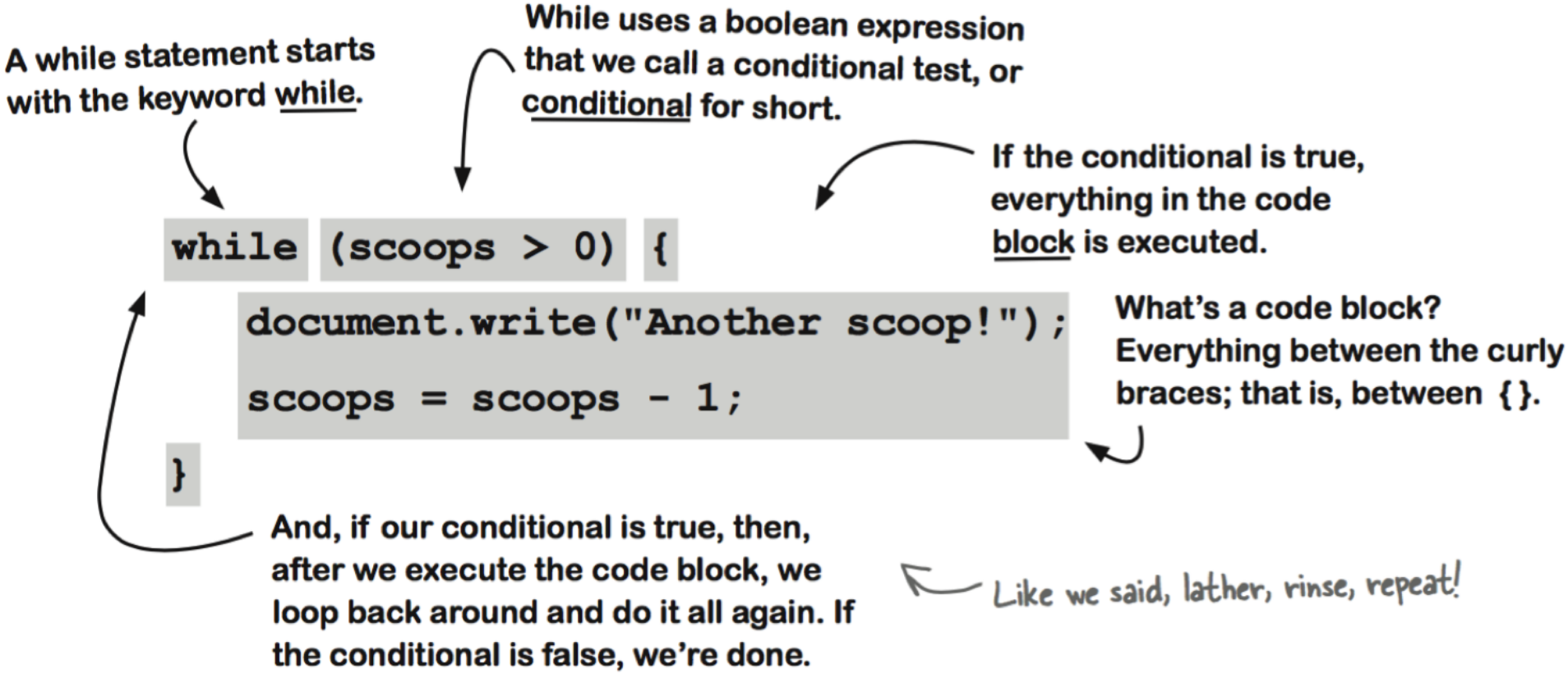
JavaScript assignment operators

```
txt1 = "What a very ";
txt1 += "nice day";
```

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

...

While statement



Example: Happy birthday (birthday.html in javascript.zip)

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Happy Birthday</title>
  </head>
  <body>
  <script>
    var name = "Joe";
    var i = 0;
    while (i < 2) {
        document.write("Happy Birthday to you.<br>");
        i = i + 1;
    }
    document.write("Happy Birthday dear " + name + ",<br>");
    document.write("Happy Birthday to you.<br>");
  </script>
</body>
</html>
```

document.write() write a string to the HTML page

Example: Happy birthday (birthday2.html in javascript.zip)

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Happy Birthday</title>
  </head>
  <body>
  <script type="text/javascript">
    var name = "Joe";
    var i = 0;
    while (i < 2) {
        console.log("Happy Birthday to you.");
        i = i + 1;
    }
    console.log("Happy Birthday dear " + name + ",");
    console.log("Happy Birthday to you.");
  </script>
</body>
</html>
```

console.log() writes a message to the console

Valori speciali

- Ad ogni variabile può essere assegnato il valore **null** che rappresenta l'assenza di un valore
- Come in SQL, **null** è un concetto diverso da *zero* (0) o *stringa vuota* (“”)
- Una variabile non inizializzata ha invece un valore indefinito **undefined**
- I due concetti si assomigliano ma non sono uguali:
  - **undefined** significa una variabile è stata dichiarata, ma non è ancora stato assegnato un valore
  - **null** è un valore di assegnazione. Esso può essere assegnato ad una variabile come una rappresentazione di valore

Tipi primitivi

- JavaScript prevede pochi tipi primitivi: **number**, **boolean**, **string** e **undefined**
- Numeri (**number**)
  - Sono rappresentati in formato floating point a 8 byte
  - Non c'è distinzione fra interi e reali
  - Esiste il valore speciale ***NaN*** (not a number) per le operazioni non ammesse (ad esempio, radice quadrata di un numero negativo)
  - Esiste il valore ***infinite*** (ad esempio, per la divisione per zero)
- Booleani (**boolean**)
  - ammettono i valori ***true*** e ***false***

Il concetto di tipo in JavaScript

- Come abbiamo detto, alle variabili non viene attribuito un tipo
  - lo assumono **dinamicamente** in base al dato a cui vengono agganciate

```
var v; // senza tipo

v = 15.7; // diventa di tipo number

v = true; // diventa di tipo boolean
```

Communicate with your user

- **Create an alert**
  - the browser gives you a quick way to alert your users through the **alert** function. Just call alert with a string containing your alert message, and the browser will give your user the message in a nice dialog box: **alert("Hello world!");**
  - Alert really should be used **only when you truly want to stop everything and let the user know something**
- **Write directly into your document**
  - Think of your web page as a document (that's what the browser calls it). You can use a function **document.write** to write arbitrary HTML and content into your page at any point. In general, this is considered **bad form**, although you'll see it used here and there

Communicate with your user (2)

- **Use the console**
  - Every JavaScript environment also has a console that can log messages from your code. To write a message to the console's log you use the function **console.log** and hand it a string that you'd like printed to the log. You can view console.log as a great tool for troubleshooting your code, but typically your users will never see your console log, so **it's not a very effective way** to communicate with them

Directly manipulate your document

- This is the way you want to be interacting with your page and users - using JavaScript you can access your actual web page, read & change its content, and even alter its structure and style! This all happens by making use of your browser's ***document object model (DOM)***. This is the best way to communicate with your user
- Using the DOM requires knowledge of **how your page is structured** and of the programming interface that is used to read and write to the page

How do I add code to my page?

- **You can place your code inline, in the <head> element**
- **You can add your code inline, in the body (just before the closing tag) of the document** When your browser loads a page, it loads everything in your page's <head> before it loads the <body>
    - if your code is in the <head>, users might have to wait a while to see the page
    - If the code is loaded after the HTML in the <body>, users will get to see the page content while they wait for the code to load

How do I add code to my page? (2)

- **Put your code in its own file and link to it from the <head>**
  - This is just like linking to a CSS file. The only difference is that you use the **src** attribute of the **<script>** tag to specify the URL to your JavaScript file
  - When your code is in an external file, it's easier to maintain (separately from the HTML) and can be used across multiple pages. But this method still has the drawback that all the code needs to be loaded before the body of the page
- **Put your code in its own file and link to it from the <body> (just before the closing tag)**
  - We have a maintainable JavaScript file that can be included in any page, and it's referenced from the bottom of the body of the page, so it's only loaded after the body of the page. But this method still has a drawback that you cannot execute the JavaScript code before the file is loaded

Loading scripts: usual approach

- Script tag with src (in head section of HTML page)

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

- Purpose
  - To define functions, objects, and variables
  - Functions will later be triggered by buttons, other user events, inline script tags with body content, etc.
- Html5 note
  - The type attribute is optional and will be omitted in future examples
- Best practice
  - Use subfolder for the JavaScript files (just as with images and CSS files)

    <script src="scripts/my-script.js"></script>

Example: code.js in javascript.zip

```
function getmessage() {
  var amount = Math.floor(Math.random() * 100000);
  var message =
    "you won $" + amount + "!\n" +
    "to collect your winnings, send your credit card\n" +
    "and bank details to oil-minister@phisher.com.";
  return(message);
}
function showwinnings1() {
  alert(getmessage());
}
function showwinnings2() {
  document.write("<h1>" + getmessage() + "</h1>");
}
```

"alert" show a dialog box with a message

"document.write" inserts text into page at location that it is called

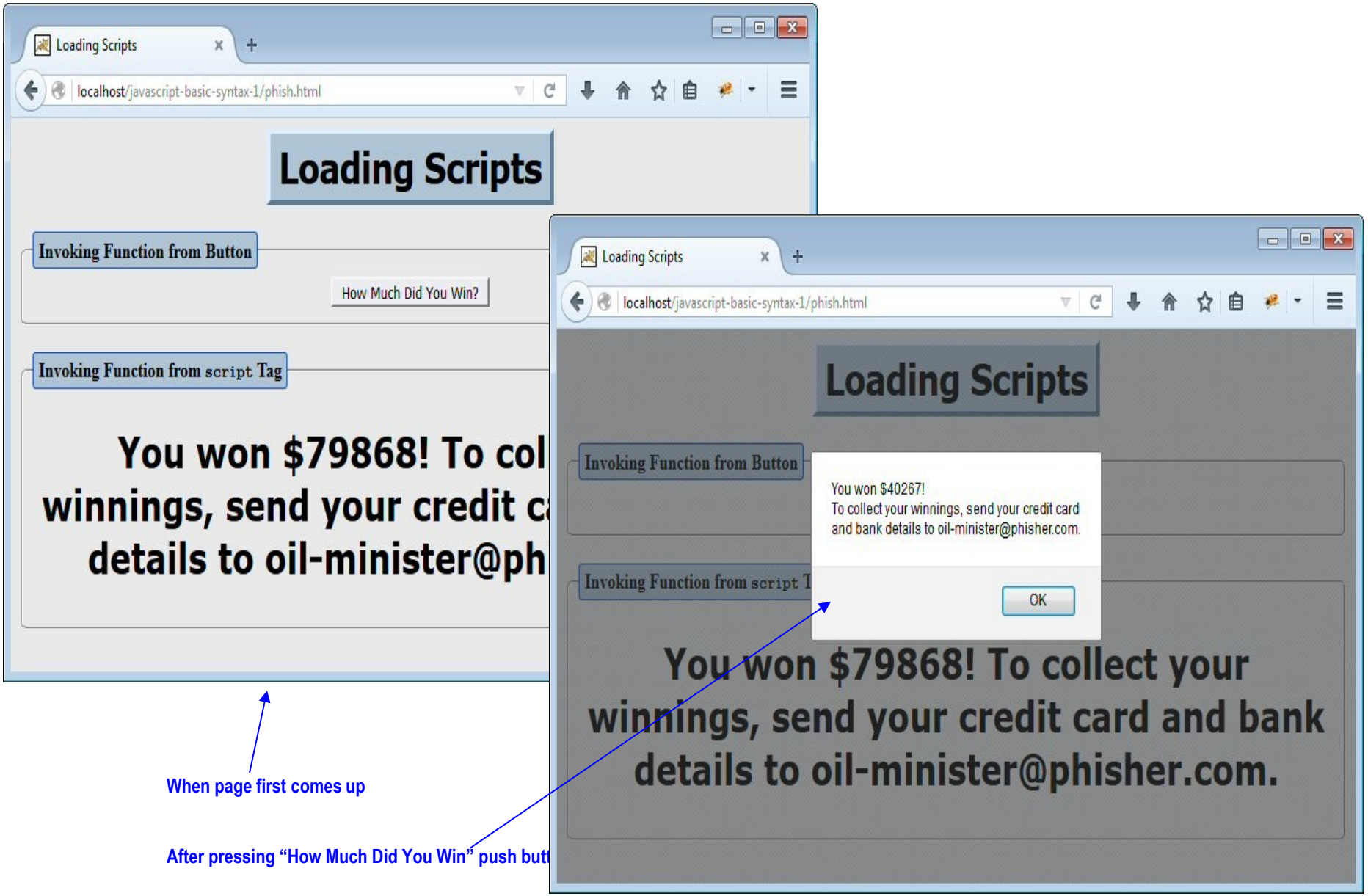
Example: win.html

```
<!doctype html>
<html>
<head><title>loading scripts</title>
...
<script src="scripts/code.js"></script>
</head>
<body>
...
   <input type="button" value="how much did you win?"
          onclick='showwinnings1()'/>
...
  <script>showwinnings2()</script>
...
</body>
</html>
```

Loads script shown on previous page

Try to move "<script src="scripts/code.js"></script> right before the body end tag.

Example (Results)

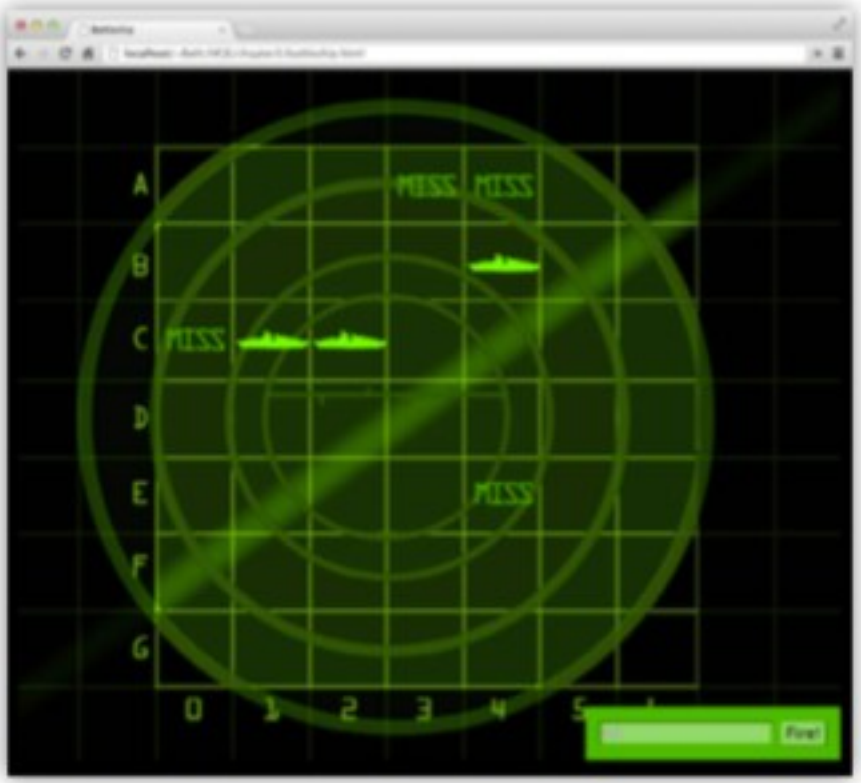


Exercise: battleship game

- **Goal:** Sink the browser's ships in the fewest number of guesses. You're given a rating, based on how well you perform
- **Setup**: When the game program is launched, the computer places ships on a virtual grid. When that's done, the game asks for your first guess
- **How you play:** *The browser will prompt you to enter a guess and you'll type in a grid location. In response to your guess, you'll see a result of "Hit", "Miss", or "You sank my battleship!" When you sink all the ships, the game ends by displaying your rating*

Simplified battleship

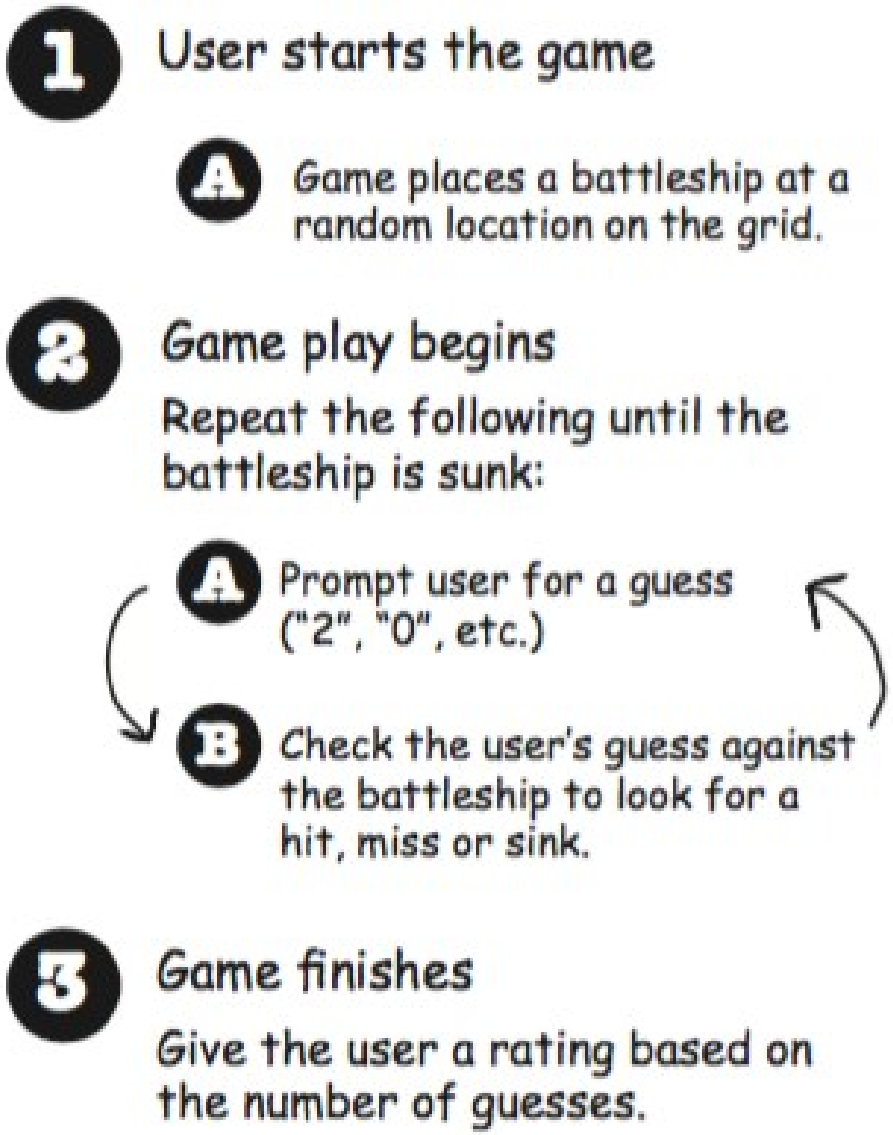
- *Objective: 7x7 graphical version with three ships, but…*
- We're going to start with a nice 1-D grid with seven locations and one ship to find



Instead of a 7x7 grid, like the one above, we're going to start with just a 1x7 grid. And, we'll worry about just one ship for now.

Notice that each ship takes up three grid locations (similar to the real board game).

High-level design



Details

- **Representing the ships**
  - Keep in mind the virtual grid
  - The user know that the battleship is hidden in three consecutive cells out of a possible seven (starting at zero), the row itself doesn't have to be represented in code
- **Getting user input**
  - Use the **prompt** function. Whenever we need to get a new location from the user, we'll use prompt to display a message and get the input, which is just a number between 0 and 6, from the user
- **Displaying the results**
  - Use alert to show the output of the game

Sample game interaction

Battleship

The HTML for the Battleship game is super simple; we just need a page that links to the JavaScript code, and that's where all the action happens.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Battleship</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Play battleship!</h1>
    <script src="battleship.js"></script>
  </body>
</html>
```

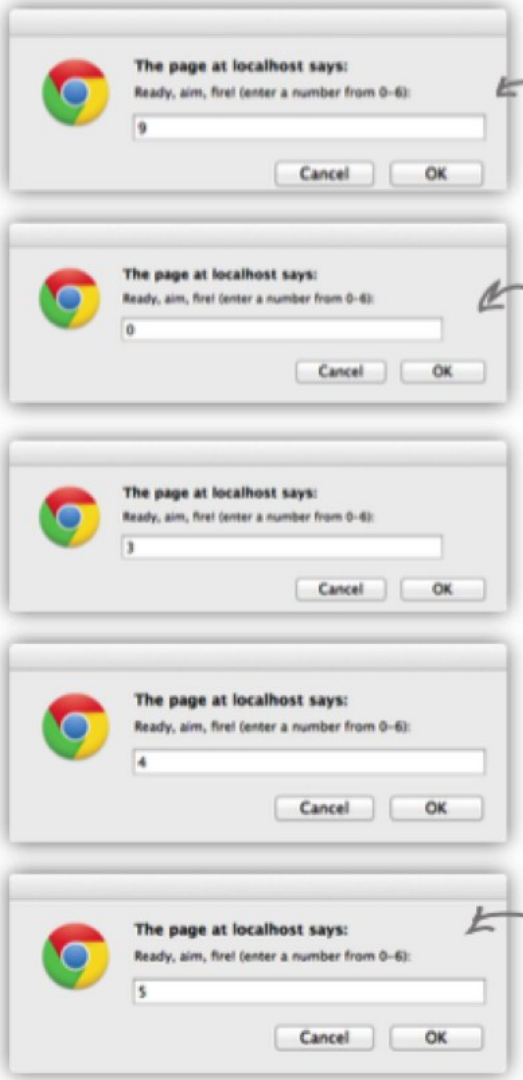
We're linking to the JavaScript at the bottom of the <body> of the page, so the page is loaded by the time the browser starts executing the code in "battleship.js".

Battleship (2)

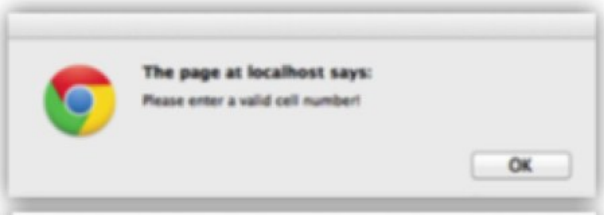
```
let pos1 = Math.floor(Math.random()*5)
let pos2 = pos1+1
let pos3 = pos1+2
let sunk = false;
let hit1 = false, hit2 = false, hit3 = false;
let guesses = 0;
let guess;
let score;
while (sunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0 to 6:)")
    if (guess < 0 || guess > 6){
        alert("Your should enter a number from 0 to 6!");
        continue;
    }
    guesses++;
    if (guess == pos1 && hit1 == false){
        hit1 = true;
        alert("Hit!");
    } else if (guess == pos2 && hit2 == false){
        hit2 = true;
        alert("Hit!");
    } else if (guess == pos3 && hit3 == false){
        hit3 = true;
        alert("Hit!");
    } else {
        alert("Missed!");
    }
    if (hit1 == true && hit2 == true && hit3 == true){
        alert("Sunk!");
        sunk = true;
    }
}
score = 3/guesses;
alert("Your score is: " + score)
```

Test it!

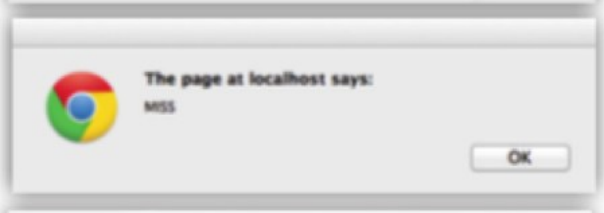
Here's what our game interaction looked like.



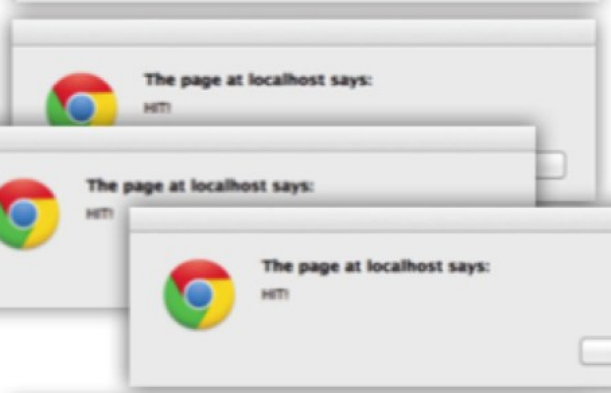
First we entered an invalid number, 9.



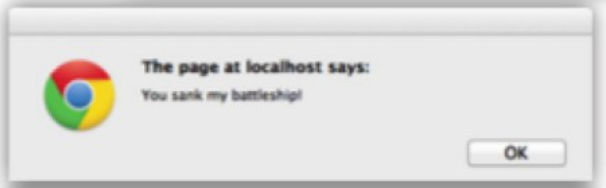
Then we entered 0, to get a miss.



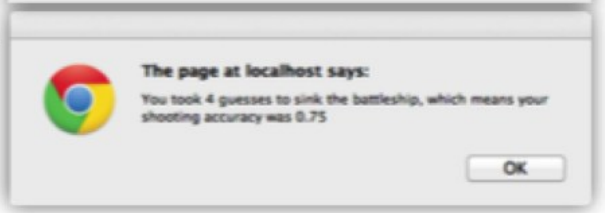
But then we get three hits in a row!



On the third and final hit, we sink the battleship.



And see that it took 4 guesses to sink the ship with an accuracy of 0.75.



Oggetti

- Gli oggetti sono tipi composti che contengono un certo numero di **proprietà** (attributi)
  - Ogni proprietà ha un **nome** e un **valore**
  - Si accede alle proprietà con l'operatore '**.**' (punto) o l'operatore '**[]**' (parentesi quadre)
  - Le proprietà non sono definite a priori: *possono essere aggiunte dinamicamente*
- Gli oggetti possono essere creati usando l'operatore **new**:

  var o = new Object()

- ***Attenzione:*** Object() è un costruttore e non una classe

Costruire un oggetto

- Un oggetto appena creato è completamente vuoto non ha né proprietà né funzioni
- Possiamo costruirlo dinamicamente
  - appena assegniamo un valore ad una proprietà, la proprietà comincia ad esistere
- Nell'esempio sottostante creiamo un oggetto e gli aggiungiamo 3 proprietà numeriche **x**, **y** e **tot**:

```
var o = new Object();
o.x = 7;
o.y = 8;
o.tot = o.x + o.y;
alert(o.tot);
```

Costanti oggetto

- Possiamo (anzi **dobbiamo!!!**) definire gli oggetti usando gli **object literal** che consistono in un elenco di attributi nella forma **nome:valore**, separati da **virgola**, racchiusi fra parentesi graffe **{ }**
  - **var/let/const nomeoggetto = {prop1:val1, prop2:val2, ... }**
  - Una prassi comune è dichiarare gli oggetto usando la keyword **const**
- Usando gli object literal creiamo un oggetto e le proprietà (valorizzate) nello stesso momento

```
var o = new Object();
o.x = 7;
o.y = 8;
o.tot = 15;
alert(o.tot);
```

Sconsigliato!

OK!

```
var o = {x:7, y:8, tot:15};
alert(o.tot);
```

Don’t use constructors!!!

- Use `{}` instead of `new Object()`
- Use `""` instead of `new String()`
- Use `0` instead of `new Number()`
- Use `false` instead of `new Boolean()`
- Use `[]` instead of `new Array()`
- Use `/()/` instead of `new RegExp()`
- Use `function (){}` instead of `new Function()`

Creare un oggetto usando la keyword “var”

```
<!DOCTYPE html>
<html>
<body>

<p>Creating a JavaScript Object.</p>

<p id="demo"></p>

<script>
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

Creating a JavaScript Objects

John is 50 years old.

- The **getElementById()** method returns the element that has the ID attribute with the specified value
- The **innerHTML** property sets or returns the HTML content (inner HTML) of an element

Creare un oggetto usando la keyword “let”

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>
<p>Creating an object:</p>

<p id="demo"></p>

<script>
let person = {
  firstName : "John",
  lastName  : "Doe",
  age       : 50,
  eyeColor  : "blue"
};

document.getElementById("demo").innerHTML = person.firstName + " " +
person.lastName;
</script>

</body>
</html>
```

Creare un oggetto usando la keyword “const”

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>
<p>Creating an object:</p>

<p id="demo"></p>

<script>
const person = {
  firstName : "John",
  lastName  : "Doe",
  age       : 50,
  eyeColor  : "blue"
};

document.getElementById("demo").innerHTML = person.firstName + " " +
person.lastName;
</script>

</body>
</html>
```

Accedere alle proprietà di un oggetto

The syntax for accessing the property of an object is:

```
objectName.property       // person.age
```

or

```
objectName["property"]    // person["age"]
```

Example: object method

```
<!DOCTYPE html>
<html>
<body>

<p>Creating and using an object method.</p>

<p>An object method is a function definition, stored as a property
value.</p>

<p id="demo"></p>

<script>
var person = {
    firstName: "John",
    lastName : "Doe",
    id       : 5566,
    fullName : function() {
       return this.firstName + " " + this.lastName;
    }
};

document.getElementById("demo").innerHTML = person.fullName();
</script>
</body>
</html>
```

Creating and using an object method.

An object method is a function definition, stored as a property value.

John Doe

Example: delete

```
<!DOCTYPE html>
<html>
<body>

<p>The delete operator deletes a property from an object.</p>

<p id="demo"></p>

<script>
var person = {
    firstname:"John",
    lastname:"Doe",
    age:50,
    eyecolor:"blue"
};
delete person.age;
document.getElementById("demo").innerHTML =
person.firstname + " is " + person.age + " years old.";
</script>

</body>
</html>
```

It applies to method objects as well

The delete operator deletes a property from an object.

John is undefined years old.

Array

- Gli array sono tipi composti i cui elementi sono accessibili mediante un indice numerico:
  - l'indice parte da zero (**0**)
  - non hanno una dimensione prefissata (simili agli ArrayList di Java)
  - espongono attributi e metodi
- Si possono dichiarare:
  - con **new Array([*dimensione*])** **(pratica sconsigliata!!!)**
  - Con gli **array literal** delimitati da **[ ]**:

    var/let/const *varname* = [val_1, val_2, ..., val_n]
  - Es. **var a = [1, 40, 22];**
  - La pratica oggi più diffusa consiste nell'usare la keyword **const**, ad es: **const a = [1, 40, 22];**
- Possono contenere elementi di tipo eterogeneo:
  - Es. **const b = [1, true, "ciao", {x:1,y:2}];**

Example: array

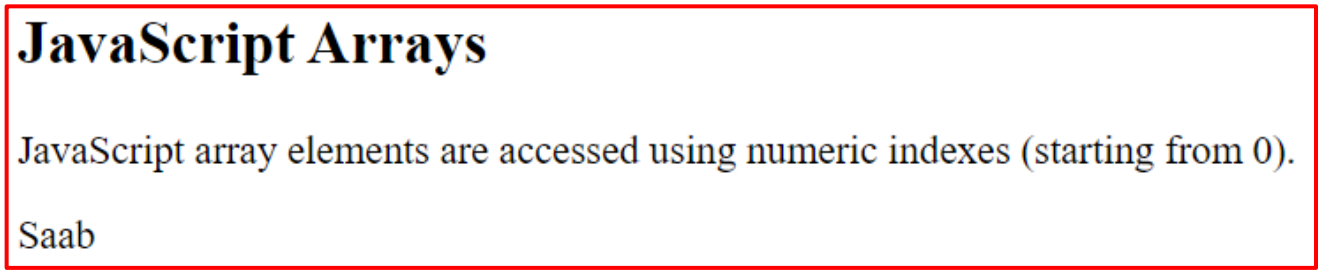
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Example: array (elements declared dynamically)

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Example: array element access

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric indexes (starting from 0).</p>

<p id="demo"></p>

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

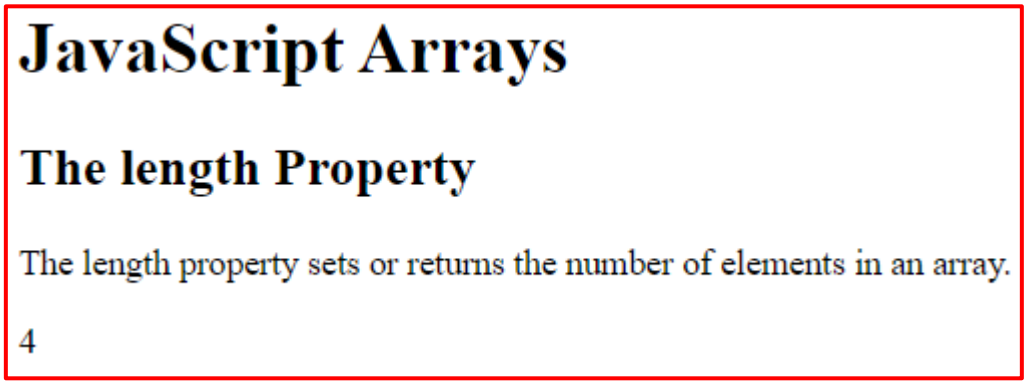


Note su Oggetti/Array dichiarata usando la keyword “const”

- Un oggetto/array dichiarato con const non può essere riassegnato
- Ma potete cambiare le proprietà degli oggetti e gli elementi degli array
- https://www.w3schools.com/js/js_const.asp

Example: array length

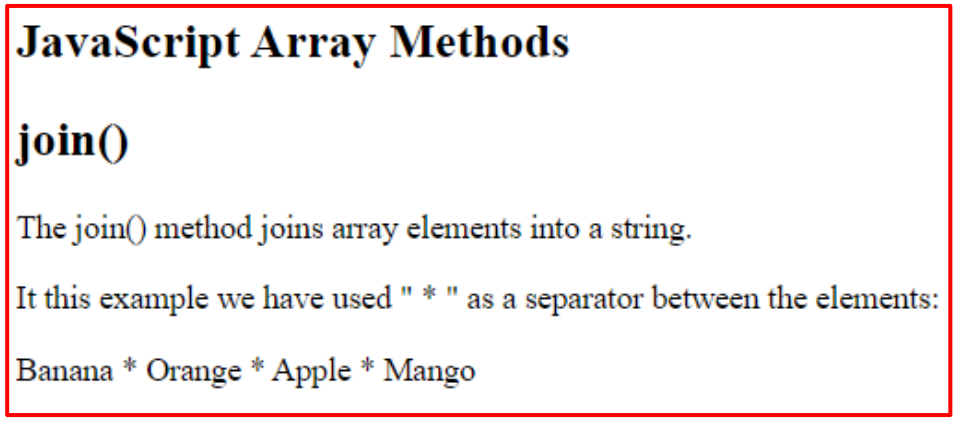
```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Arrays</h1>
<h2>The length Property</h2>

<p>The length property sets or returns the number of elements in an array.</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length =  fruits.length;

document.getElementById("demo").innerHTML = length;
</script>

</body>
</html>
```



Array Methods

- https://www.w3schools.com/js/js_array_methods.asp
- Ad es.: toString(), join(), …

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>
<h2>join()</h2>
<p>The join() method joins array elements into a string.</p>
<p>It this example we have used " * " as a separator between the elements:</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>

</body>
</html>
```



Example: in

```
<!DOCTYPE html>
<html>
<body>

<p>The in operator returns true if the specified property is in the
specified object.</p>

<p id="demo"></p>

<script>
// Arrays
var cars = ["Saab", "Volvo", "BMW"];
// Objects
var person = {firstName:"John", lastName:"Doe", age:50};

document.getElementById("demo").innerHTML =
    ("Saab" in cars) + "<br>" +
    (0 in cars) + "<br>" +
    (1 in cars) + "<br>" +
    (4 in cars) + "<br>" +
    ("length" in cars) + "<br>" +

    ("firstName" in person) + "<br>" +
    ("age" in person) + "<br>" +

    // Predefined objects
    ("PI" in Math) + "<br>" +
    ("NaN" in Number) + "<br>" +
    ("length" in String);
</script>

</body>
</html>
```

The in operator returns true if the specified property is in the specified object.

false
true
true
false
true

true
true

true
true
true

Example: instanceof

```
<!DOCTYPE html>
<html>
<body>

<p>The instanceof operator returns true if the specified object is an
instance of the specified object.</p>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];

document.getElementById("demo").innerHTML =
    (cars instanceof Array) + "<br>" +
    (cars instanceof Object) + "<br>" +
    (cars instanceof String) + "<br>" +
    (cars instanceof Number);
</script>

</body>
</html>
```

The instanceof operator returns true if the specified object is an instance of the specified object.

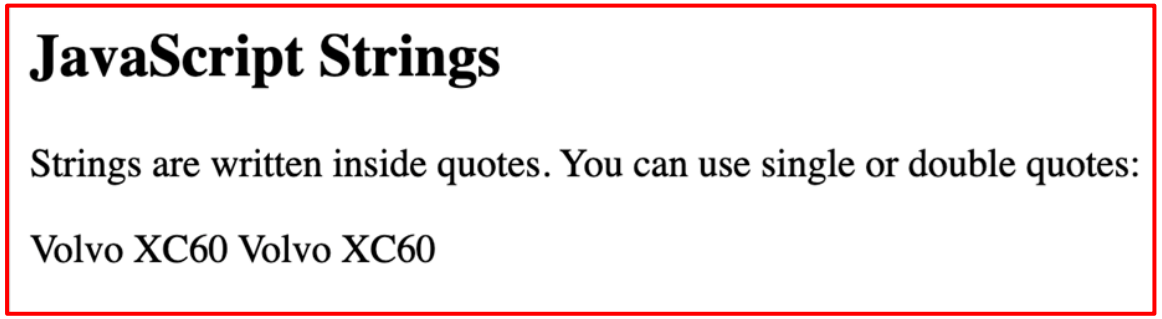
true
true
false
false

Stringhe

- Sono sequenze arbitrarie di caratteri in formato UNICODE a 16 bit
- Esiste la possibilità di definire le stringhe tramite **string literal** delimitate da apici singoli (**'ciao'**) o doppi (**"ciao"**)
- È possibile la concatenazione con l'operatore **+**
- Espongono proprietà e metodi

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Strings are written inside quotes. You can use single or double quotes:
</p>

<p id="demo"></p>

<script>
let carName1 = "Volvo XC60"; // Double quotes
let carName2 = 'Volvo XC60'; // Single quotes

document.getElementById("demo").innerHTML =
carName1 + " " + carName2;
</script>

</body>
</html>
```



Stringhe come oggetti?

- Le stringhe in JavaScript non sono oggetti
  - `“ciao” instanceof Object // returns false!!!`
- Possiamo però invocare metodi su una stringa o accedere ai suoi attributi così come facciamo per gli oggetti
- Possiamo infatti scrivere
  - `let s = “ciao”;`
  - `let n = s.length; // to get the string length`
  - `let t = s.charAt(1); // to get the second char`

https://www.w3schools.com/jsref/jsref_obj_string.asp

String and number basics

- You can use double or single quotes

```
let names = ["joe", 'jane', "john", 'juan'];
```

- Numbers can be converted to strings
  - Conversion via toString()

```
let text = num.toString();
```

  - Automatic conversion during concatenations

```
let val = 3 + "abc" + 5;  // result is "3abc5"
```

- Number conversion with fixed precision

```
let n = 123.4567;

let val = n.toFixed(2); //result is 123.46 (not 123.45)
```

String and number basics (2)

- Strings can be compared with ==

```
"foo" == 'foo' // returns true
```

- Strings can be converted to numbers

```
let i = parseInt("37.9 blah");
// result is 37 – ignores blah


let d = parseFloat("6.02 blah");
// result is 6.02 – ignores blah
```

- Number conversion from float to integer:

```
let d = parseInt(6.02);
// result is 6
```

Core string methods

- Simple methods
  - charAt, indexOf, lastIndexOf, substring, toLowerCase, toUpperCase, trim…

```
"Hello".charAt(1); // returns "e"
"Hello".indexOf("l");
// returns 2 (if no match, returns -1)
"hello".substring(1,3); // returns "el"
"Hello".toUpperCase(); // "HELLO"
```

- Methods that use regular expressions
  - match, replace, search, split, …

https://www.w3schools.com/jsref/jsref_obj_string.asp

The typeof operator

- In JavaScript there are 5 different data types that can contain values:
  - string
  - number
  - boolean
  - object
  - function
- There are 6 types of objects:
  - Object
  - Date
  - Array
  - String
  - Number
  - Boolean
- And 2 data types that cannot contain values:
  - null
  - undefined

The typeof operator (2)

- You can use the JavaScript typeof operator to find the data type (returned as a string) of the operand

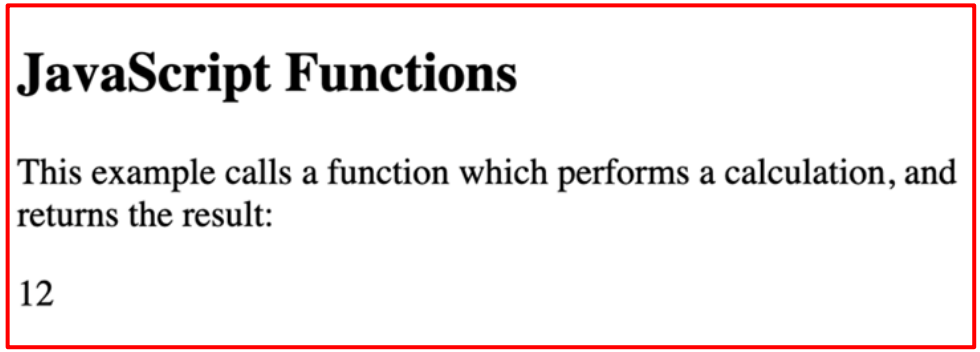
```
typeof "John"                    // Returns "string"
typeof 3.14                      // Returns "number"
typeof NaN                       // Returns "number"
typeof false                     // Returns "boolean"
typeof [1,2,3,4]                 // Returns "object"
typeof {name:'John', age:34}     // Returns "object"
typeof new Date()                // Returns "object"
typeof function () {}            // Returns "function"
typeof myCar                     // Returns "undefined"
typeof null                      // Returns "object"
```

Funzioni

- Una funzione è un frammento di codice JavaScript che viene definito una volta e usato in più punti
  - Ammette zero o più parametri che sono privi di tipo
  - Può restituire un valore il cui tipo non viene definito
  - La mancanza di tipo è coerente con la scelta fatta per le variabili
- Le funzioni possono essere definite utilizzando la parola chiave **function**

Esempio: funzioni

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a calculation, and returns
the result:</p>

<p id="demo"></p>

<script>
function myFunction(p1, p2) {
  return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>

</body>
</html>
```



Function literal

- Esistono le **function literal** che permettono di definire una funzione e poi di assegnarla ad una variabile con una sintassi un po' inusuale:

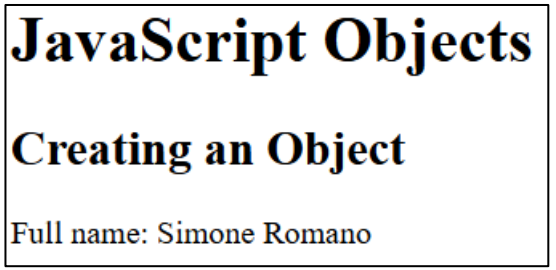
```
var sum =
  function(x,y) { return x+y; }
```

- Es: **sum(4, 3)** ritorna il valore 7

Function literal (2)

- Le function literal tornano utili quando si vuole aggiunger un metodo ad un oggetto estistente

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Objects</h1>
<h2>Creating an Object</h2>

<p id="demo"></p>

<script>
const person = {name:"Simone", surname:"Romano"};

person.fullName = function (){
    return this.name + " " + this.surname;
};

document.getElementById("demo").innerHTML =
"Full name: " + person.fullName();
</script>

</body>
</html>
```



JavaScript is pass-by-value

- JavaScript passes arguments to a function using ***pass-by-value***. What that means is that each argument is *copied* into the parameter variable

1 Let's declare a variable age, and initialize it to the value 7.

```
var age = 7;
```

2 Now let's declare a function addOne, with a parameter named x, that adds 1 to the value of x.

```
function addOne(x) {
    x = x + 1;
}
```

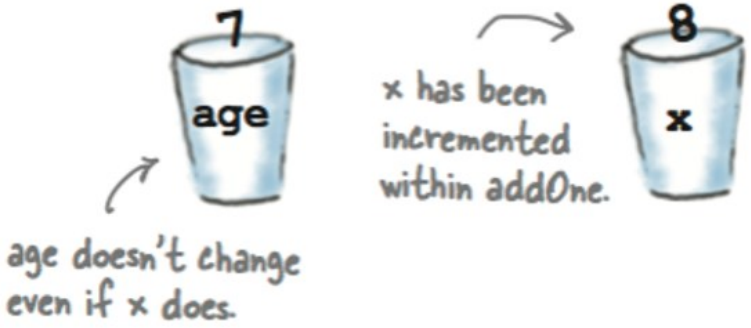
3 Now let's call the function addOne, pass it the variable age as the argument. The value in age is copied into the parameter x.

```
addOne(age);
```

4 Now the value of x is incremented by one. But remember x is a copy, so only x is incremented, not age.

We're incrementing x.

```
function addOne(x) {
  x = x + 1;
}
```



Passing objects to functions

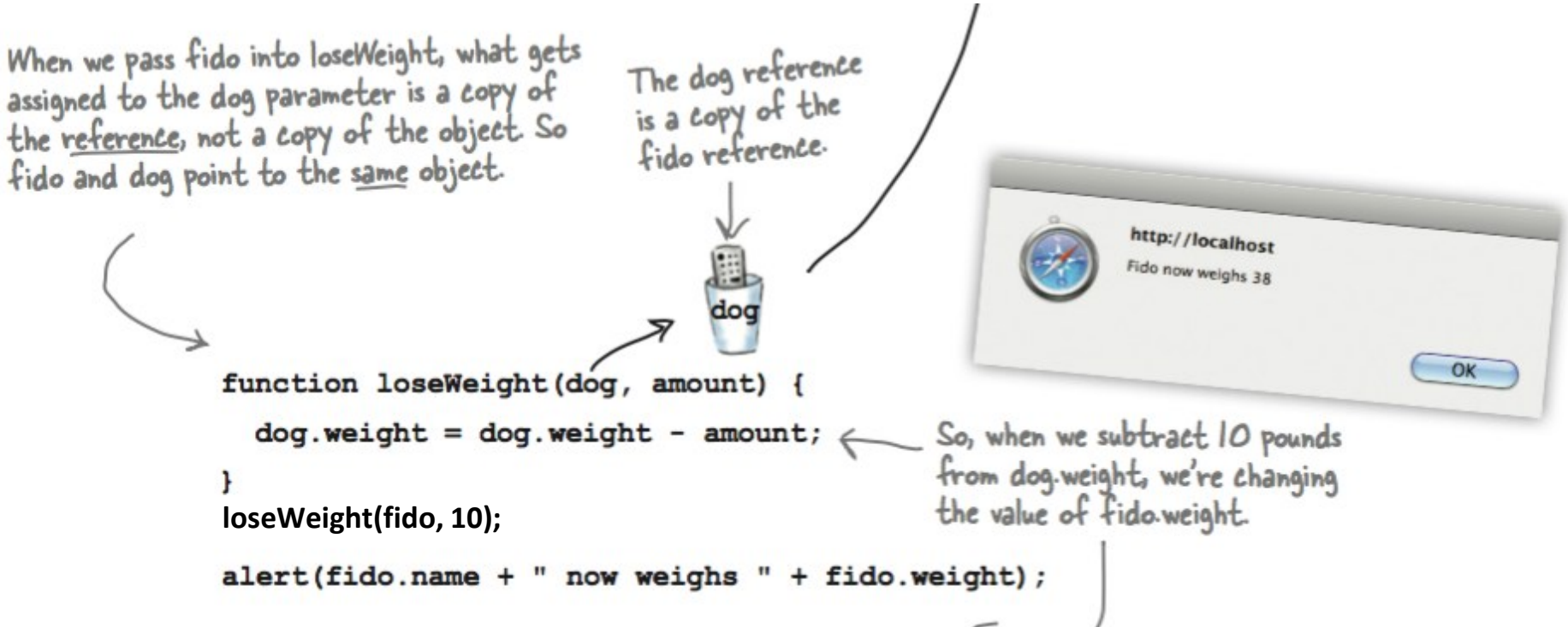
- When an object is assigned to a variable, that variable holds a ***reference*** to the object, not the object itself
- When you call a function and pass it an object as argument, you're passing **the *object reference*,** not the object itself

- So using our pass by value semantics, a copy of the reference is passed into the parameter, and that reference remains a pointer to the original object
- **if you change a property of the object in a function, you're changing the property in the *original* object**

Putting Fido on a diet....

- The dog parameter of the loseWeight function gets a copy of the reference to fido. So any changes to the properties of the parameter variable affect the object that was passed in:

const fido = {name: “Fido”, weight: 48}



This

- Quando una funzione viene assegnata ad una proprietà di un oggetto viene chiamata metodo dell'oggetto (Object Method)
- La cosa è possibile perché, come abbiamo visto, una funzione può essere assegnata ad una variabile
  - In questo caso all'interno della funzione si può utilizzare la parola chiave **this** per accedere all'oggetto di cui la funzione è una proprietà

```
<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
```

The JavaScript *this* Keyword

In this example, **this** refers to the **person** object.

Because **fullName** is a method of the person object.

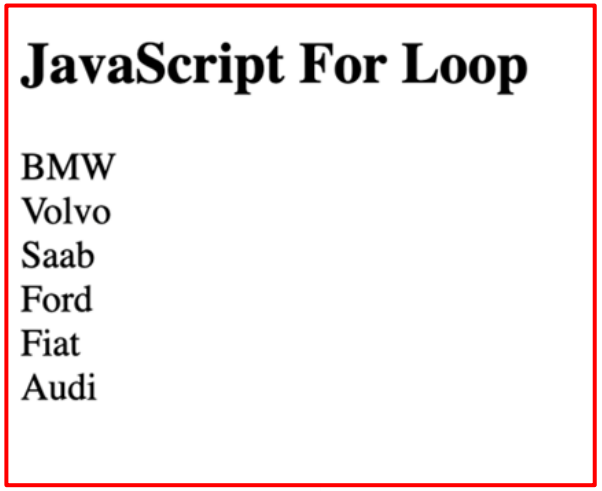
John Doe

Istruzioni

- Un programma JavaScript è una sequenza di istruzioni
- Buona parte delle istruzioni JavaScript hanno la stessa sintassi di C e Java
- Si dividono in:
  - Espressioni (uguali a Java): assegnamenti, invocazioni di funzioni e metodi, ecc.
  - Istruzioni composte: blocchi di istruzioni delimitate da parentesi graffe (uguali a Java)
  - Istruzione vuota: punto e virgola senza niente prima
  - Istruzioni etichettate: normali istruzioni con un'etichetta davanti (sintassi: ***label: statement***)
  - Strutture di controllo: if, for, while, ecc.
  - Definizioni e dichiarazioni: var, let, const, function
  - Istruzioni speciali: break, continue, return

For statement

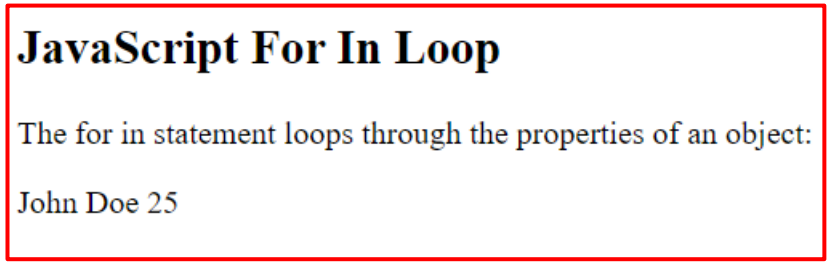
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For Loop</h2>

<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

let text = "";
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```



For statement (2)

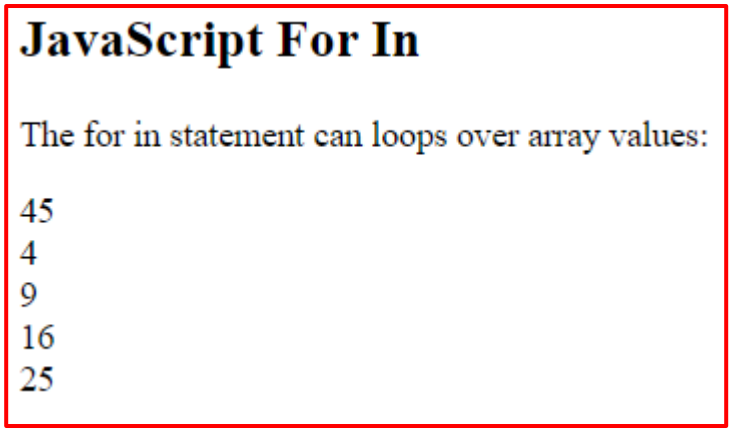
- La struttura **for/in** permette di scorrere le proprietà di un oggetto (e quindi anche un array) con la sintassi: **for (*variable* in *object*)**

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For In Loop</h2>
<p>The for in statement loops through the properties of an
object:</p>

<p id="demo"></p>

<script>
const person = {fname:"John", lname:"Doe", age:25};

let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}

document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```



Esempio: for-in in un array

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For In</h2>
<p>The for in statement can loops over array values:</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];

let txt = "";
for (let x in numbers) {
  txt += numbers[x] + "<br>";
}

document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```



Array operations

- In JavaScript, arrays have methods

```
const nums = [1,2,3];
nums.reverse(); // -> [3,2,1]
[1,2,3].reverse(); // -> [3,2,1]
```

- Most important methods:
  - toString, join, forEach, reverse
  - push, pop, concat
  - sort
  - map
  - filter
  - slice, splice
  - indexOf

Push, pop, concat

- Push

```
const nums = [1,2,3];
nums.push(4);
nums; // -> [1,2,3,4]
```

- Pop

```
nums; // -> [1,2,3,4]
const val = nums.pop();
val; // -> 4
nums; // -> [1,2,3]
```

- Concat

```
nums; // -> [1,2,3]
const nums2 = nums.concat([4,5,6]);
nums2; // -> [1,2,3,4,5,6]
```

Sort

- With no arguments (default comparisons)
  - Note the odd behavior with numbers: they are sorted lexicographically, not numerically

```
["hi","bye","hola","adios"].sort();
 // -> ["adios","bye","hi","hola"]

[1,-1,-2,10,11,12,9,8].sort();
 // -> [-1,-2,1,10,11,12,8,9]
```

array.sort does a lexicographic sort by default, for a numeric sort, provide your own function

```
function compareNumbers(a, b) {
    return a - b;
}
sort(compareNumbers)
```

```
[1,-1,-2,10,11,12,9,8].sort(compareNumbers);
 // -> [-2,-1,1,8,9,10,11,12]
```

Filter

- Calls function on each element, keeps only the results that "pass" (return true for) the test. Returns new array; does not modify original array
- Example

```
function isEven(n) { return(n % 2 == 0); }
[1,2,3,4].filter(isEven); // -> [2,4]
```

More array methods

- Slice
  - Returns sub-array

```
[9,10,11,12].slice(0, 2); // -> [9,10]
[1,2,3].slice(0); // -> [1,2,3] (i.e., makes
copy of array)
```

- Indexof
  - Finds index of matching element

```
[9,10,11].indexOf(10); // -> 1
[9,10,11].indexOf(12); // -> -1
```

Splice

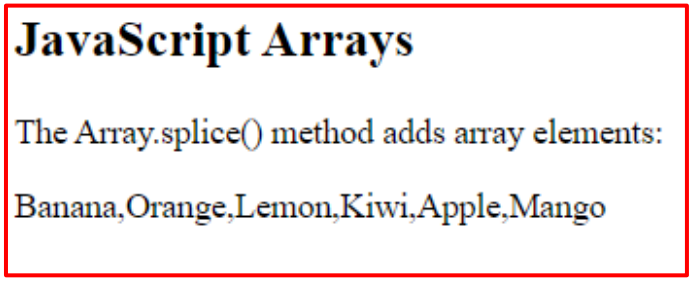
- The **splice** method can be used to add new items to an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.splice(2, 0, "Lemon", "Kiwi");
```

- The first parameter (2) defines the position **where** new elements should be **added** (spliced in)
- The second parameter (0) defines **how many** elements should be **removed**
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**

Example: splice

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The Array.splice() method adds array elements:</p>

<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];

// At position 2, add 2 elements:
fruits.splice(2, 0, "Lemon", "Kiwi");

document.getElementById("demo").innerHTML = fruits;
</script>

</body>
</html>
```



Use splice to remove elements from an array:
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_splice2

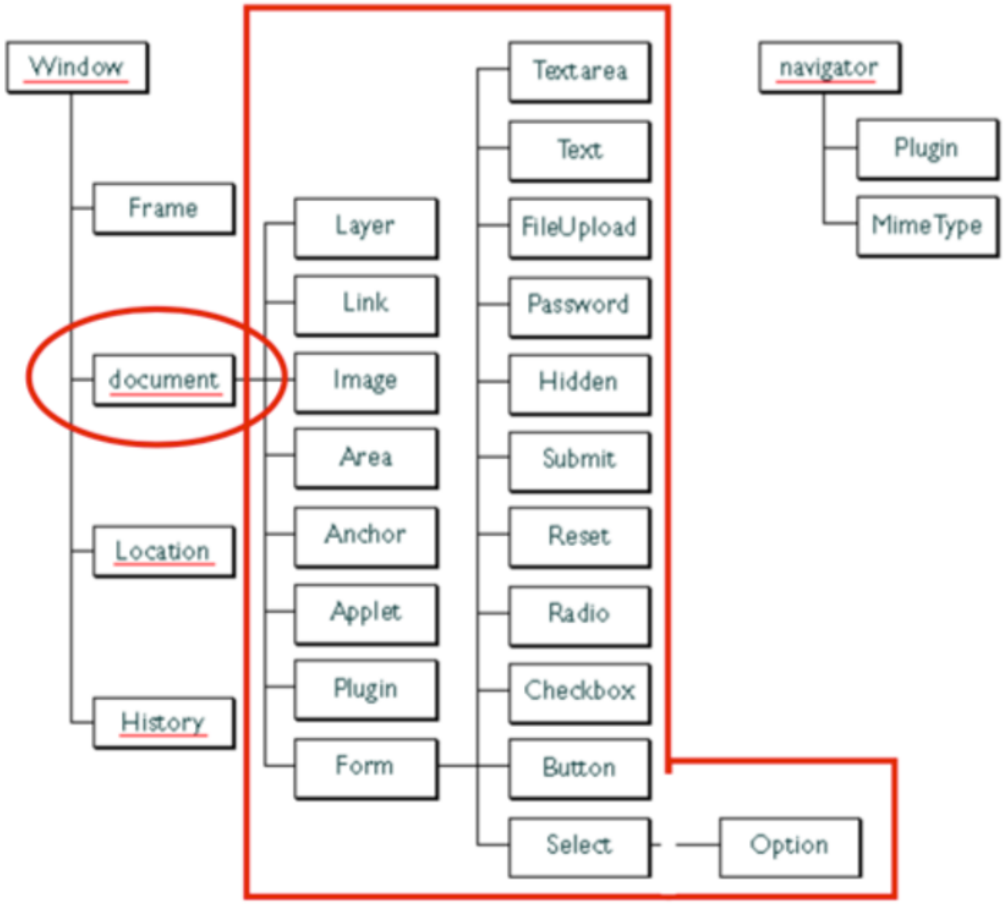
L’oggetto globale e funzioni predefinite

- In JavaScript esiste un oggetto **globale implicito**
- Questo oggetto espone alcune funzioni predefinite:
  - **eval(expr)** valuta la stringa expr (che contiene un’espressione JavaScript)
  - **isFinite(number)** dice se il numero è finito
  - **isNaN(number)** dice se il numero è NaN
  - **parseInt(str])** converte la stringa str in un intero
  - **parseFloat(str)** converte la stringa str in un numero
  - …

Browser objects

- Per interagire con la pagina HTML, JavaScript utilizza una gerarchia di oggetti predefiniti denominati Browser Objects e DOM Objects

La gerarchia che ha come radice document corrisponde al DOM



Rilevazione del browser

- Per accedere ad informazioni sul browser si utilizza l'oggetto **navigator** che espone una serie di proprietà:

Proprietà	Descrizione
appCodeName	Nome in codice del browser (poco utile)
appName	Nome del browser (es. Microsoft Internet Explorer)
appVersion	Versione del Browser (es. 5.0 (Windows))
cookieEnabled	Dice se i cookies sono abilitati
platform	Piattaforma per cui il browser è stato compilato (es. Win32)
userAgent	Stringa passata dal browser come header user-agent (es. "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1;)") È possibile esplorare la proprietà userAgent per mobile browser quali iPhone, iPad, o Android.

```
<html>
  <body>
    <script>
      document.write('Hello '+navigator.appName+'!<br>');
      document.write('Versione: '+navigator.appVersion+'<br>');
      document.write('Piattaforma: '+navigator.platform);
    </script>
  </body>
</html>
```

Hello Netscape!
Version 5.0 (Macintosh; Intel Mac OS X 10_14_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1 Safari/605.1.15
Piattaforma MacIntel

Rilevazione delle proprietà dello schermo

- L'oggetto **screen** permette di ricavare informazioni sullo schermo
- **screen** espone alcune utili proprietà tra cui segnaliamo **width** e **height** che permettono di ricavarne le dimensioni

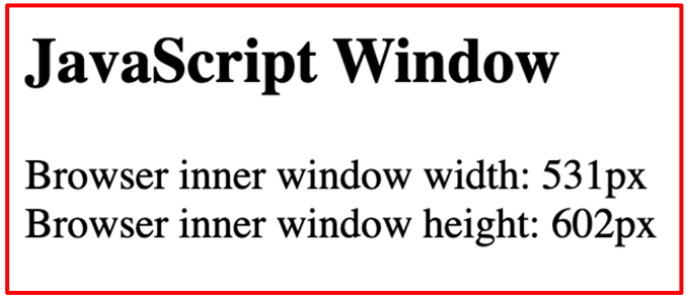
```
<html>
  <body>
    <script>
      document.write('Schermo:
          '+screen.width+'x'+screen.height+' pixel<br>');
    </script>
  </body>
</html>
```

Schermo: 1360x768 pixel

Window size

- Two properties can be used to determine the size of the browser **window**
- Both properties return the sizes in pixels:
  - **window.innerHeight** - the inner height of the browser window (in pixels)
  - **window.innerWidth** - the inner width of the browser window (in pixels)
- A practical JavaScript solution (covering all browsers):

```
<script>
document.getElementById("demo").innerHTML =
"Browser inner window width: " + window.innerWidth + "px<br>" +
"Browser inner window height: " + window.innerHeight + "px";
</script>
```



Modello ad eventi ed interattività

- Per avere una reale interattività bisogna utilizzare il meccanismo degli eventi
- JavaScript consente di associare script agli eventi causati dall'interazione dell'utente con la pagina
  - L'associazione avviene mediante attributi collegati agli elementi della pagina HTML
- Gli script prendono il nome di gestori di eventi (**event handlers**)
- Nelle risposte agli eventi si può intervenire sul DOM modificando dinamicamente la struttura della pagina (DHTML):

  DHTML = JavaScript + DOM + CSS

Events

- An HTML event can be something the browser does, or something a user does
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
  - …
- Often, when events happen, you may want to do something
- JavaScript lets you execute code when events are detected
- HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements

Gestori di evento

- Per agganciare un gestore di evento ad un evento si utilizzano gli attributi degli elementi HTML
- La sintassi è:

  <tag eventHandler="JavaScript code">
- Esempio:

  <input type="button" value="Calculate” **onClick=‘alert("Calcolo")’/>**

  - *Attenzione. È necessario alternare doppi apici e apice singolo*

- *È possibile inserire più istruzioni in sequenza, ma è meglio definire delle funzioni*

Events-1

https://www.w3schools.com/jsref/dom_obj_event.asp

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Events-2

Form events:

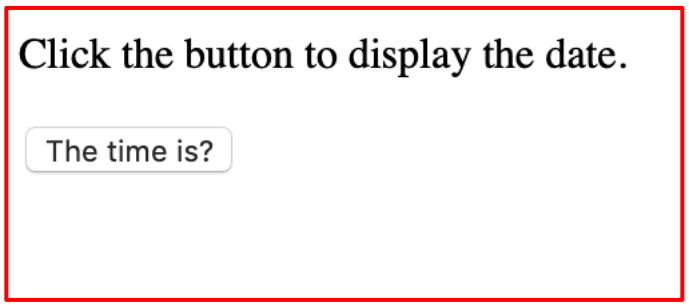
Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

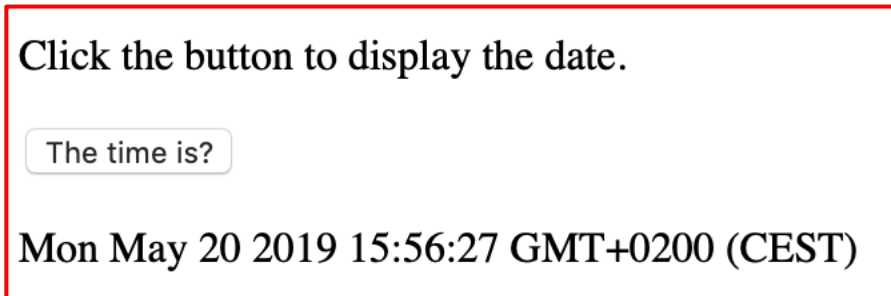
Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Example: onClick event

```
<!DOCTYPE html>
<html>
<body>
        <p>Click the button to display the date.</p>
        <button onclick="displayDate()">The time is?</button>

<script>
        function displayDate() {
            document.getElementById("demo").innerHTML = Date();
        }
</script>

        <p id="demo"></p>
</body>
</html>
```

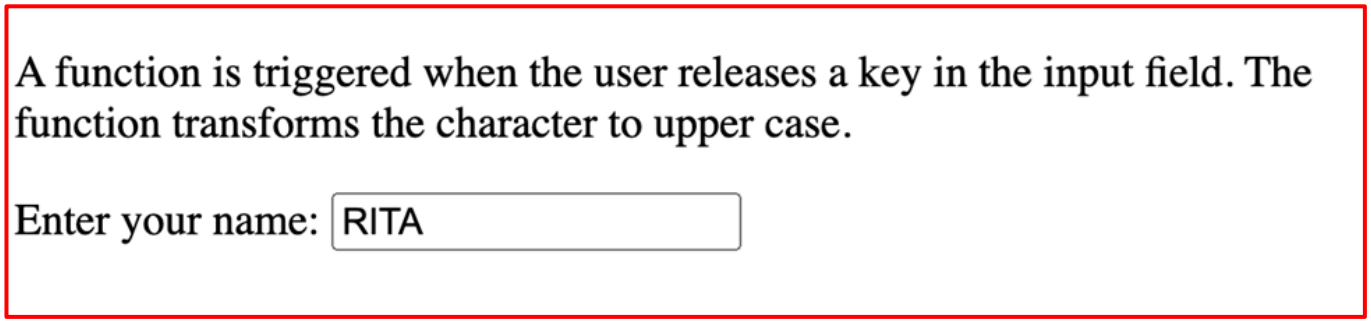




https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onclick

Example: onKeyUp event

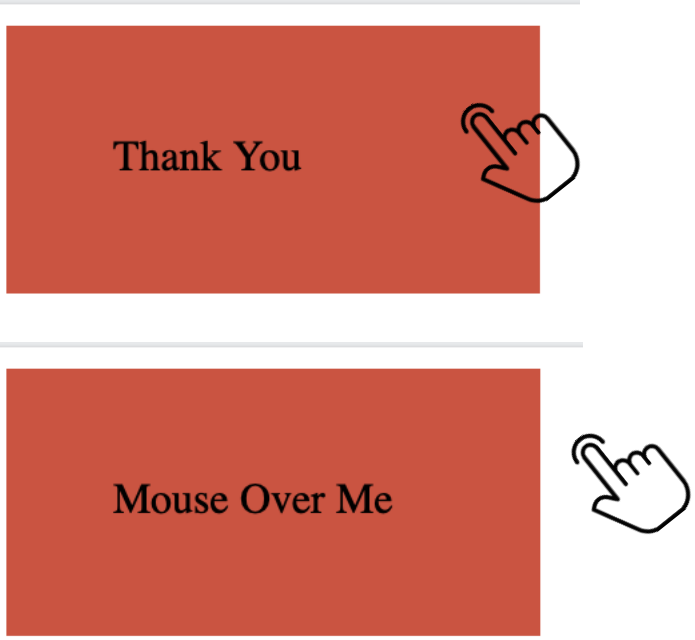
```
<!DOCTYPE html>
<html>
<body>

<p>A function is triggered when the user releases a key in the
input field. The function transforms the character to upper
case.</p>
Enter your name: <input type="text" id="fname"
onkeyup="myFunction()">

<script>
function myFunction() {
  let x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>

</body>
</html>
```



https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onkeyup

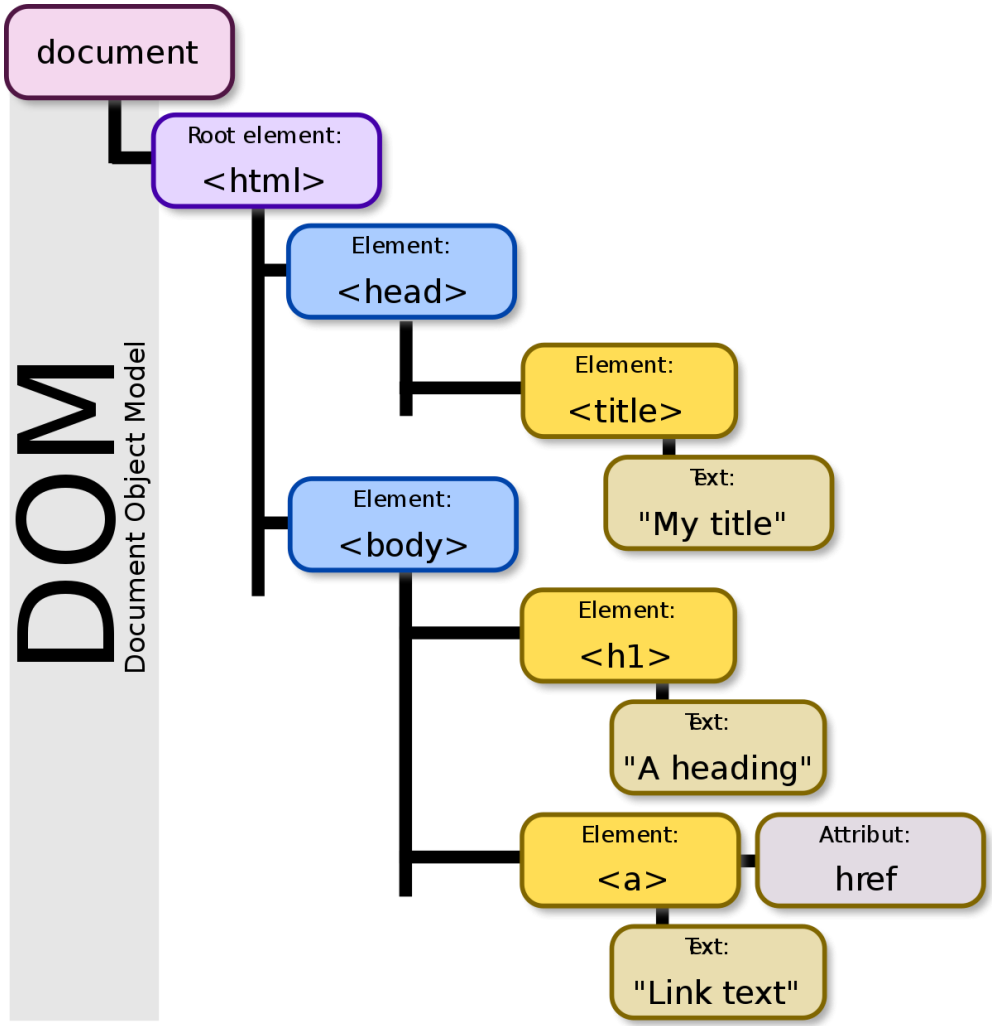
Example: onMouseOver and onMouseOut events

```
<!DOCTYPE html>
<html>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
    obj.innerHTML="Thank You"
}

function mOut(obj) {
    obj.innerHTML="Mouse Over Me"
}
</script>

</body>
</html>
```



The DOM

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>My blog</title>
  <script src="blog.js"></script>
</head>
<body>
  <h1>My blog</h1>
  <div id="entry1">
    <h2>Great day bird watching</h2>
    <p>
      Today I saw three ducks!
      I named them
      Huey, Louie, and Dewey.
    </p>
    <p>
      I took a couple of photos...
    </p>
  </div>
</body>
</html>
```



Esplorare il DOM: document

- Il punto di partenza per accedere al Documento Object Model (DOM) della pagina è l'oggetto **document**
- **document** espone, tramite le sue proprietà, alcune collezioni di oggetti HTML, tra cui:
  - **forms[]** - Returns all <form> elements
  - **images[]** - Returns all <img> elements
  - **links[]** - Returns all <area> and <a> elements that have a href attribute
- L'accesso agli elementi delle collezioni può avvenire per indice (ordine di definizione nella pagina) o per id (attributo id dell'elemento):

```
document.links[0]
document.links["myId"]
```

- Esempi di accesso a tali elementi:

```
document.links[0].href //->http://www.unisa.it
document.links["myID"].href  //->http://www.unisa.it
```

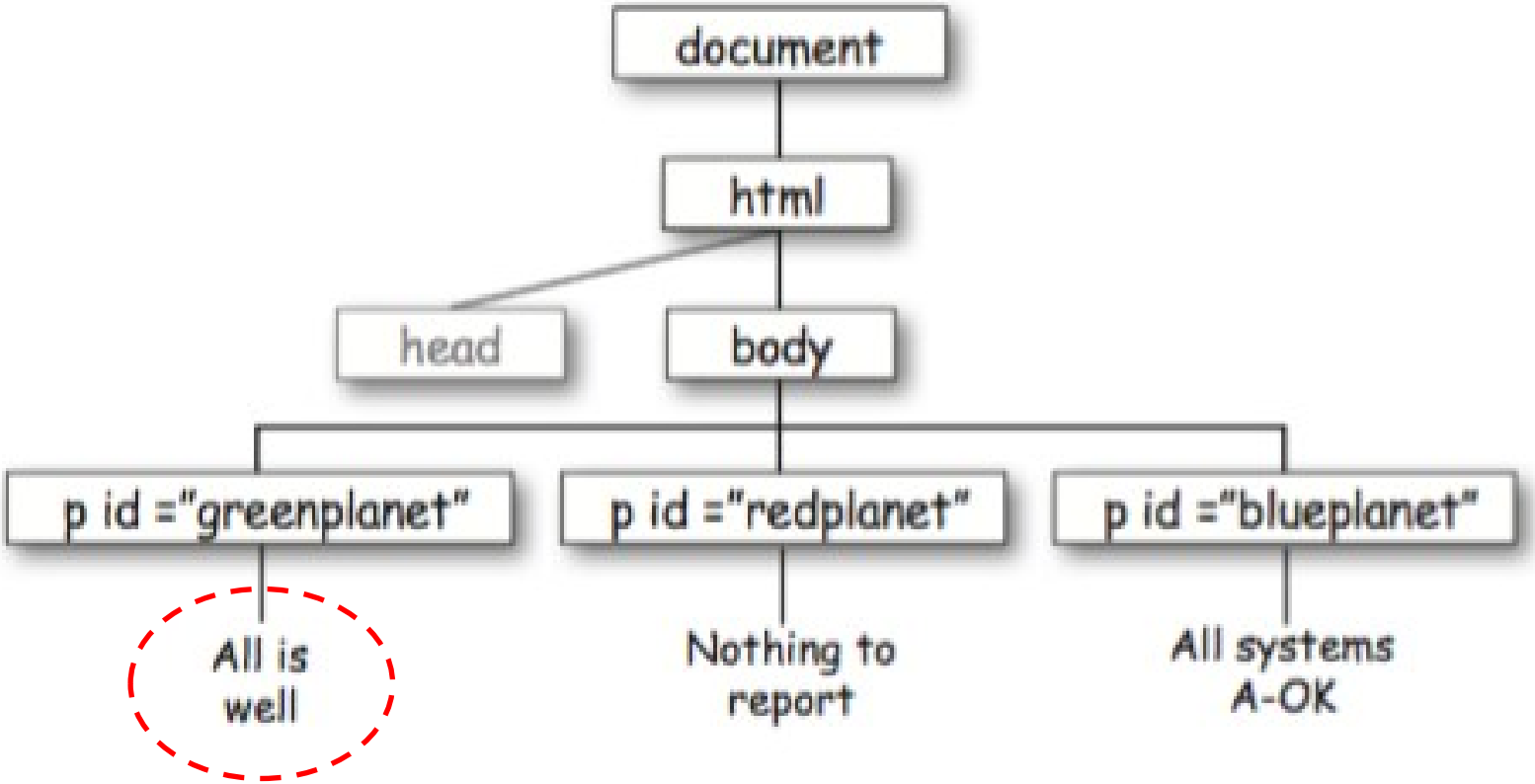
Document

- Metodi:
  - **write()**: scrive del testo nel documento
  - **writeln()**: come write() ma aggiunge un a capo
  - …
- Proprietà:
  - **lastModified**: data e ora di ultima modifica
  - **cookie**: tutti i cookies associati al documento
    - rappresentati da una stringa di coppie *nome-valore* separate da “;”
  - **title**: titolo del documento
  - **URL**: url del documento
  - …

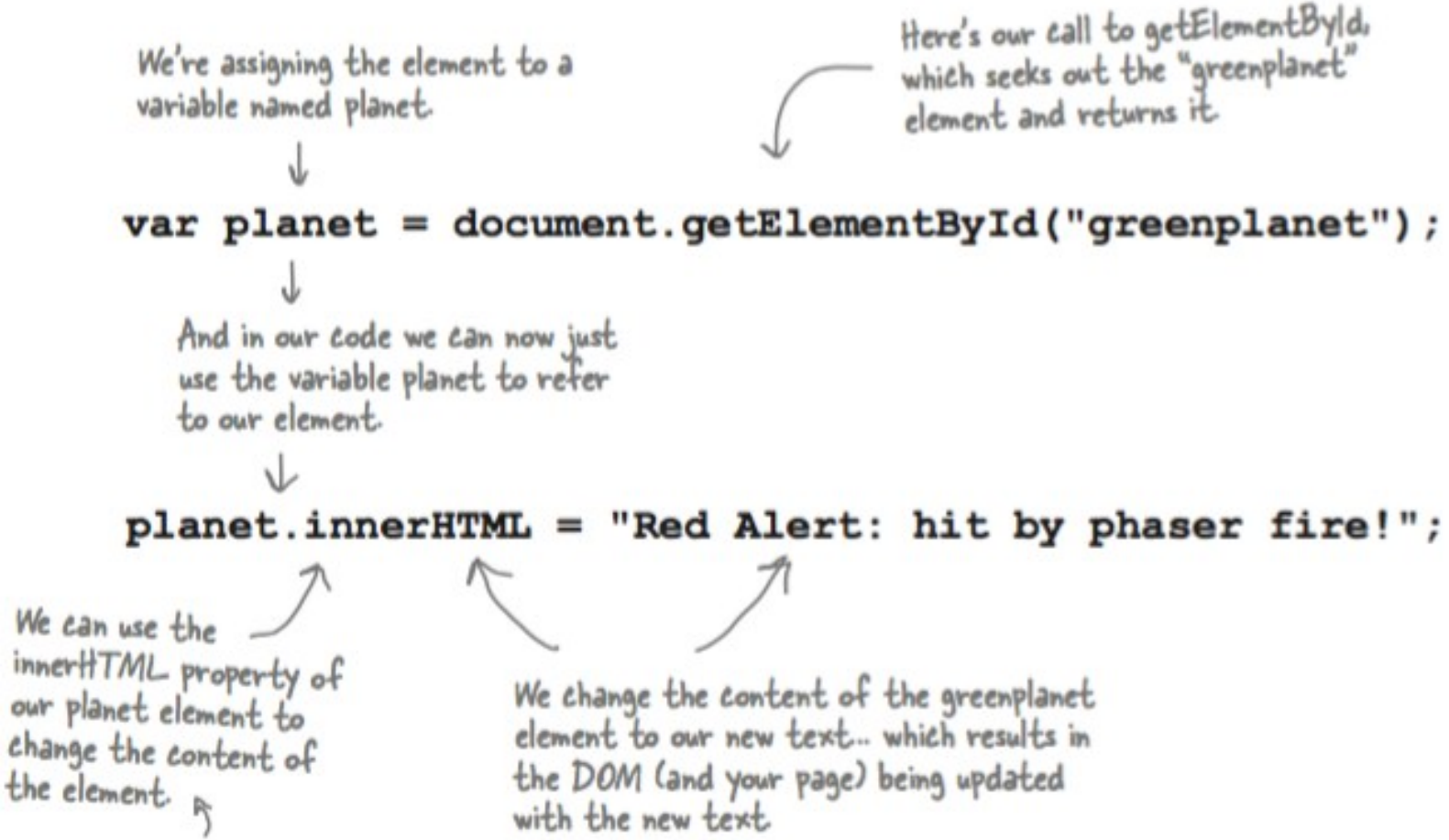
Document (2)

- **getElementById()** returns the element that has the ID attribute with the specified value
- **getElementsByClassName()** returns an array containing all elements with the specified class name
- **getElementsByTagName()** returns an array containing all elements with the specified tag name
- **querySelectorAll()** returns an array of elements that match a specified CSS selector

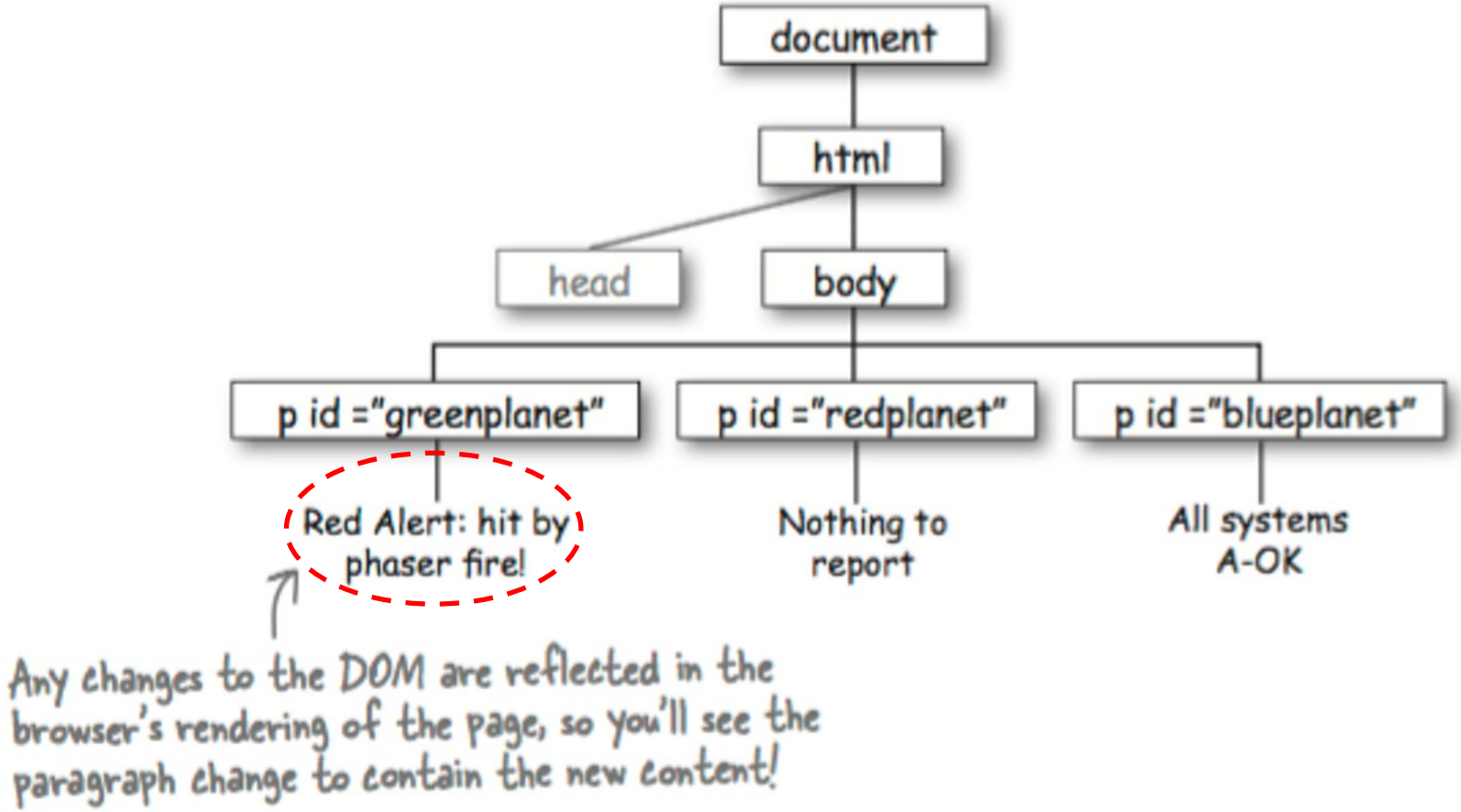
Example: change the DOM



Example: use getElementByID



Example: DOM has been changed



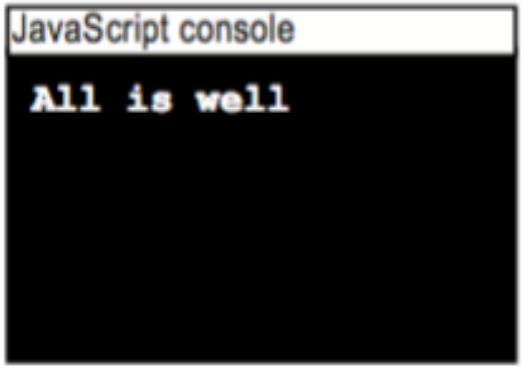
Finding the inner HTML

- The innerHTML property is an important property that we can use to read or replace the content of an element

```
var planet = document.getElementById("greenplanet");
console.log(planet.innerHTML);
```

We're just passing the planet.innerHTML property to console.log to log to the console.

The content of the innerHTML property is just a string, so it displays just like any other string in the console.



Changing the inner HTML

```
var planet = document.getElementById("greenplanet");
planet.innerHTML = "Red Alert: hit by phaser fire!";
console.log(planet.innerHTML);
```

Now we're changing the content of the element by setting its innerHTML property to the string "Red Alert: hit by phaser fire!"

So when we log the value of the innerHTML property to the console we see the new value.

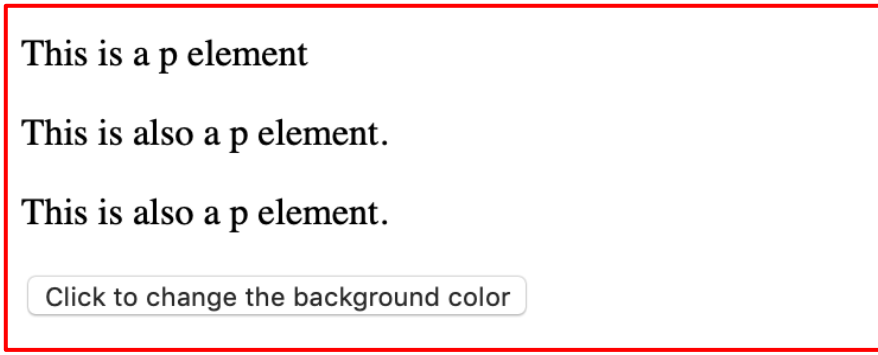
JavaScript console

```
Red Alert: hit by
phaser fire!
```

And the web page changes too!

Example: getElementsByTagName method

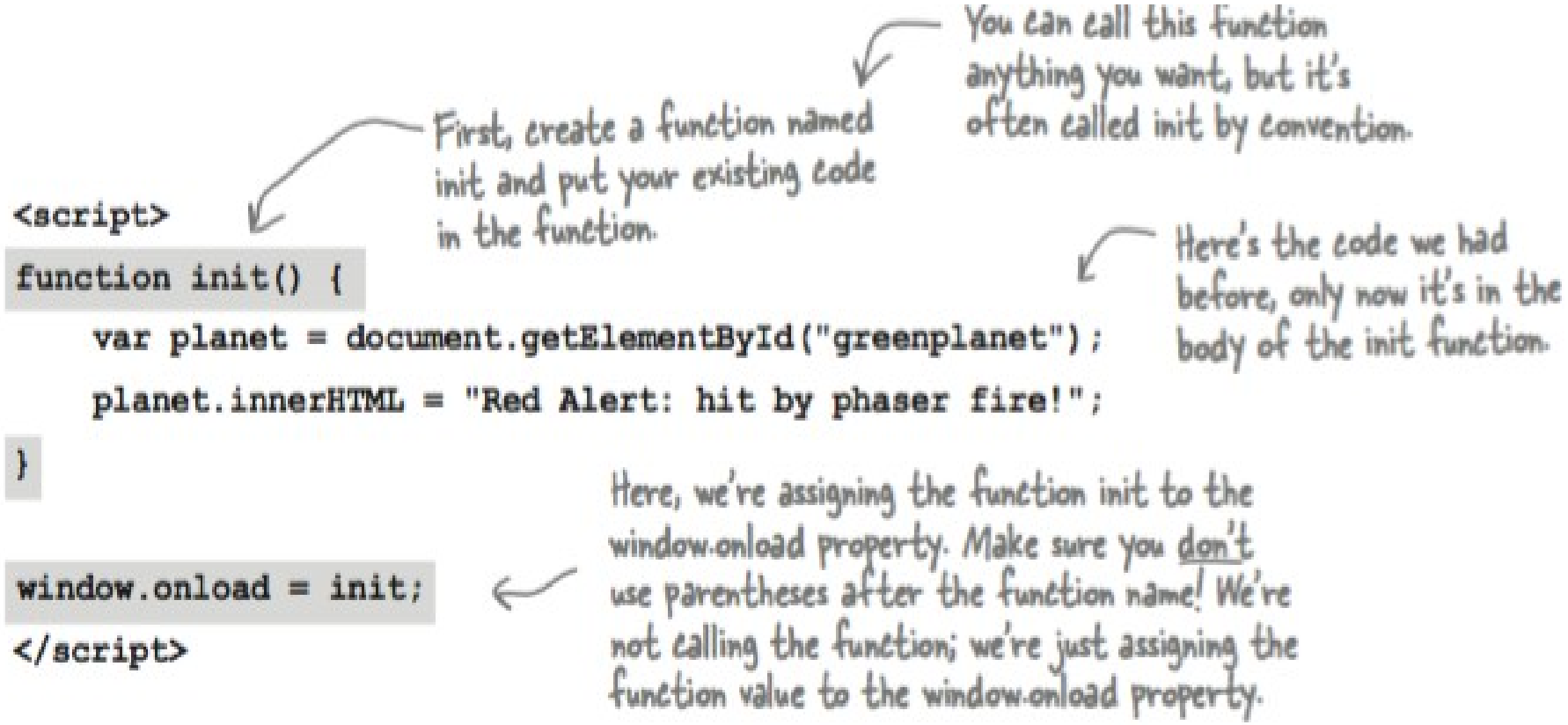
```
<!DOCTYPE html>
<html>
<body>

<p>This is a p element</p>

<p>This is also a p element.</p>

<p>This is also a p element.</p>

<button onclick="myFunction()">Click to change the background color</button>

<script>
function myFunction() {
  var x = document.getElementsByTagName("P");
  for (var i = 0; i < x.length; i++) {
    x[i].style.backgroundColor = "red";
  }
}
</script>

</body>
</html>
```



Important!

- **When you are changing the DOM, consider that some elements could not be loaded yet.** If you try change elements that are not loaded yet, your changes will not affect these elements.
- In such a situation, you would like to execute your code once the page is fully loaded. To do so:
  - First **create a function** that has the code you would like to execute once the page is fully loaded
  - Next, **take** the window object, and assign the function to its onload property

onLoad (the “page is loaded” event)



Example: onload on the body tag

NO

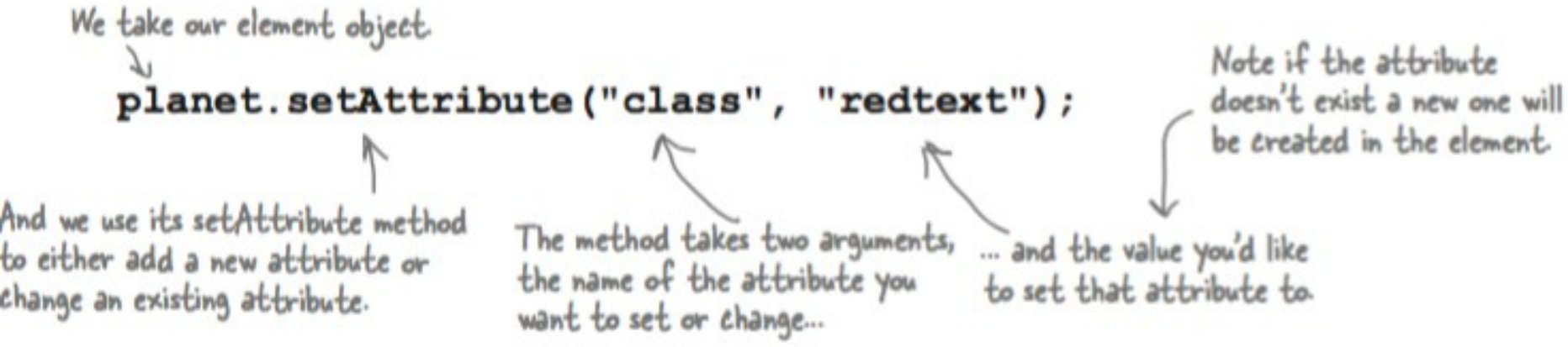
```
<!DOCTYPE html>
<html>
<body>
<h1>Hello World!</h1>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "loaded...";
}

myFunction();
</script>

<p id="demo">...</p>
</body>
</html>
```

OK

```
<!DOCTYPE html>
<html>
<body onload="myFunction()">

<h1>Hello World!</h1>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "loaded...";
}
</script>

<p id="demo">...</p>
</body>
</html>
```

How to set an attribute with setAttribute

- Element objects have a **setAttribute** method that you can call to set the value of an HTML element's attribute if it already exists, or to add a new attribute if the attribute does not exist



- It is possible to add a style attribute with a value to an element, but it is not recommended because it can overwrite other properties in the style attribute

Bad performance, not recommended!

```
element.setAttribute("style", "background-color: green;");
```

```
element.style.backgroundColor = "green";
```

OK

Example

```
<head>
  <meta charset="utf-8">
  <title>Planets</title>
  <style>
    .redtext { color: red; }
  </style>
  <script>
    function init() {
        var planet = document.getElementById("greenplanet");
        planet.innerHTML = "Red Alert: hit by phaser fire!";
        planet.setAttribute("class", "redtext");
    }
    window.onload = init;
  </script>
</head>
```

Remember: bad performance, not recommended!

We've got the redtext class included here so when we add "redtext" as the value for the class attribute in our code, it turns the text red.

And to review: we're getting the greenplanet element, and stashing the value in the planet variable. Then we're changing the content of the element, and finally adding a class attribute that will turn the text of the element red.

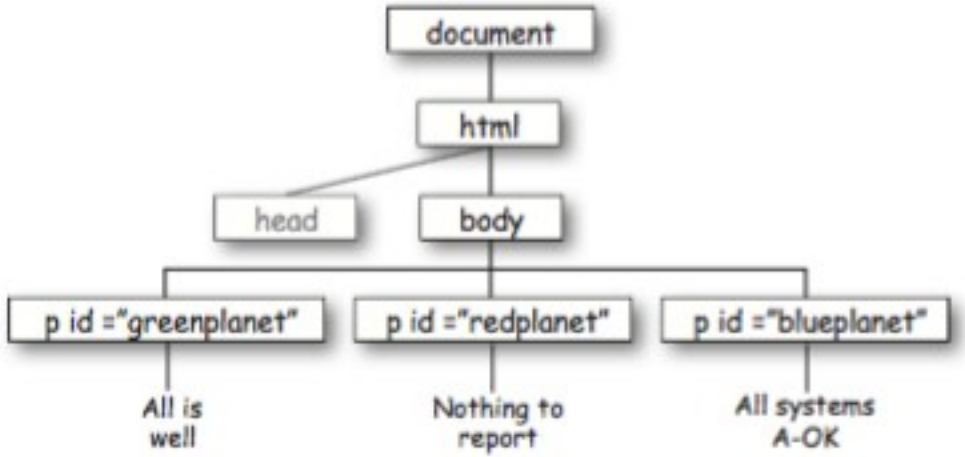
We're calling the init function only when the page is fully loaded!

```
planet.className = "redtext";
```

OK

Before...

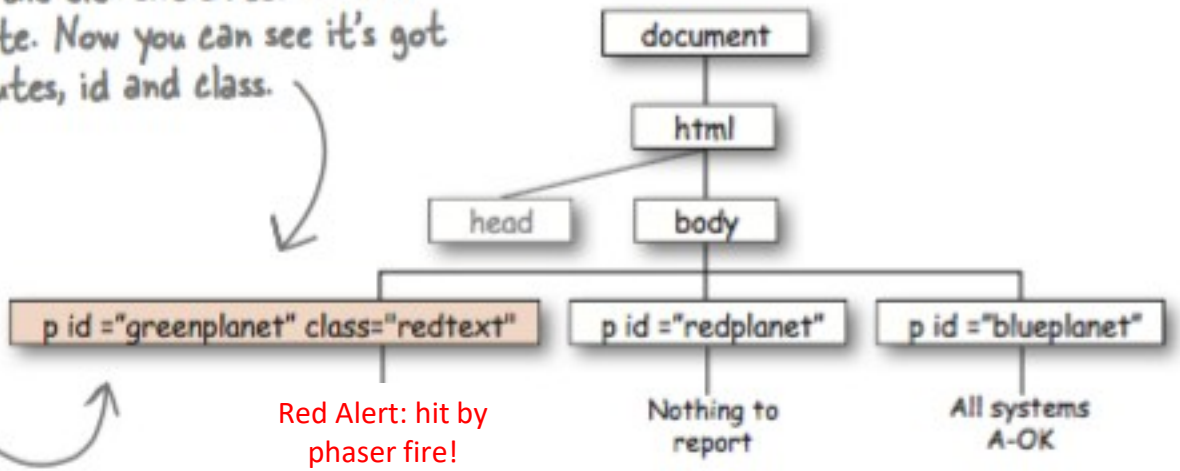
Here's the element before we call the setAttribute method on it. Notice this element already has one attribute, id.



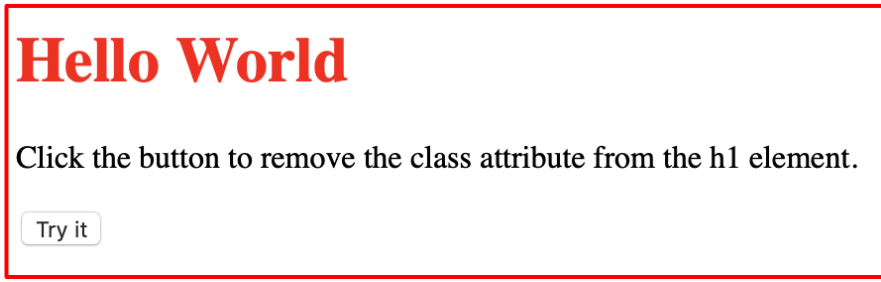
And After

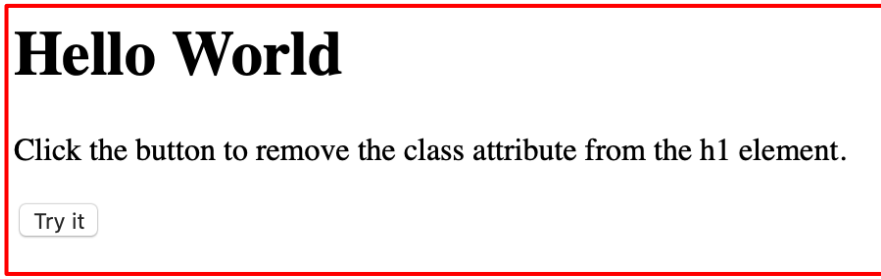
And here's the element after we call setAttribute. Now you can see it's got two attributes, id and class.

Remember, when we call the setAttribute method, we're changing the element object in the DOM, which immediately changes what you see displayed in the browser.



How to delete an attribute with removeAttribute

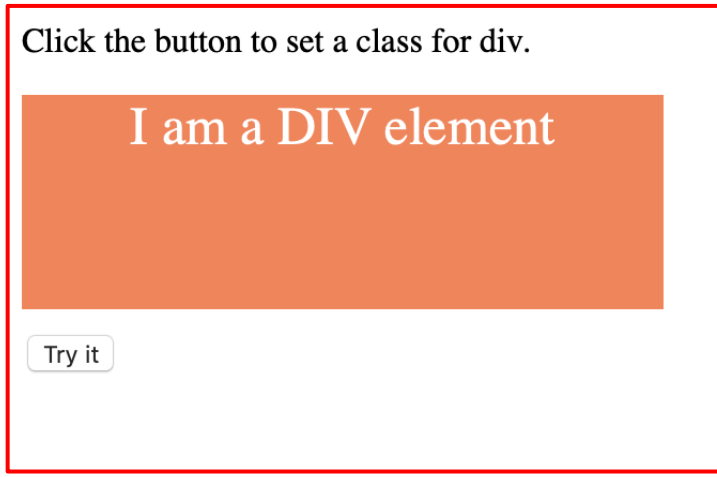





```
<!DOCTYPE html>
<html>
<head>
<style>
.democlass { color: red; }
</style>
</head>
<body>

<h1 class="democlass">Hello World</h1>

<p id="demo">Click the button to remove the class attribute from the h1
element.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementsByTagName("h1")[0].removeAttribute("class");
}
</script>

</body>
</html>
```

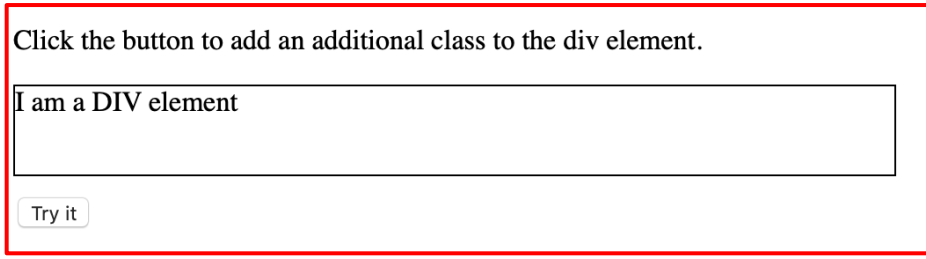
JavaScript can change the **class** attribute

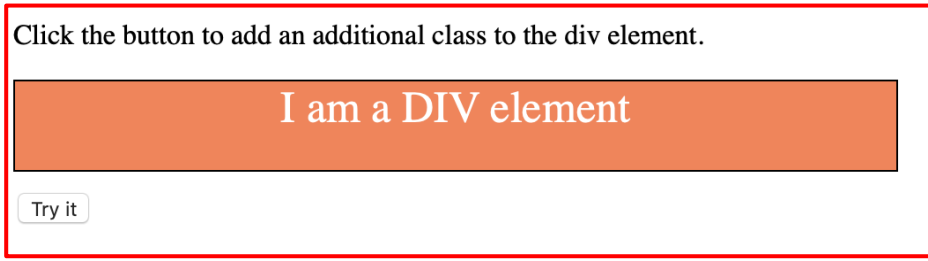
```
<!DOCTYPE html>
<html>
<head>
<style>
.mystyle {
    width: 300px;
    height: 100px;
    background-color: coral;
    text-align: center;
    font-size: 25px;
    color: white;
    margin-bottom: 10px;
}
</style>
</head>
<body>

<p>Click the button to set a class for div.</p>

<div id="myDIV">I am a DIV element</div>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
    document.getElementById("myDIV").className = "mystyle";
}
</script>

</body>
</html>
```



JavaScript can add a **class**

```
<!DOCTYPE html>
<html>
<head>
<style>
.mystyle {
  width: 500px; height: 50px;
  border: 1px solid black; margin-bottom: 10px;
}

.anotherClass {
  background-color: coral; text-align:
  center; font-size: 25px; color: white;
}
</style>
</head>
<body>

<p>Click the button to add an additional class to the div element.</p>

<div id="myDIV" class="mystyle">
    I am a DIV element
</div>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("myDIV").className += " anotherClass";
}
</script>

</body>
</html>
```





JavaScript can change CSS styles

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<p>Changing the HTML style:</p>


<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>


</body>
</html>
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_change_style

DOM is good for…

Get elements from the DOM

- Use document.getElementById…
- Use tag names, class names to retrieve not just one element, but a whole set of elements (say all elements in the class “on_sale”)
- Get form values the user has typed in, like the text of an input element
- …

Create and add elements to the DOM

- You can create new elements and you can also add those elements to the DOM. Of course, any changes you make to the DOM will show up immediately as the DOM is rendered by the browser

DOM is good for… (2)

Remove elements from the DOM

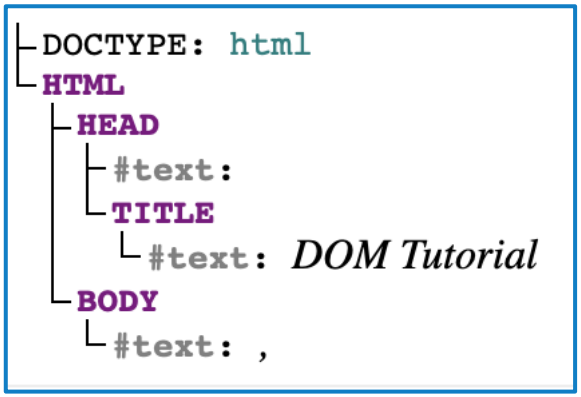
- You can also remove elements from the DOM by taking a parent element and removing any of its children
- You'll see the element removed in your browser window as soon as it is deleted from the DOM

Traverse the elements in the DOM

- Once you have a handle to an element, you can find all its children, you can get its siblings (all the elements at the same level), and you can get its parent. The DOM is structured just like a family tree!

Navigating between nodes

- You can use the following node properties to navigate between nodes with JavaScript:
  - **parentNode**
  - **childNodes[nodenumber]**
  - **firstChild**
  - **lastChild**
  - **nextSibling**
  - **previousSibling**



- ***Warning!!!***
  - A common error in DOM processing is to expect an element node to contain text
  - In this example: **<title>DOM Tutorial</title>**, the element node <title> does not contain text. It contains a text node with the value "DOM Tutorial"
  - The value of the text node can be accessed by the node's **innerHTML** property, or the **nodeValue**

Example: access with nodeValue

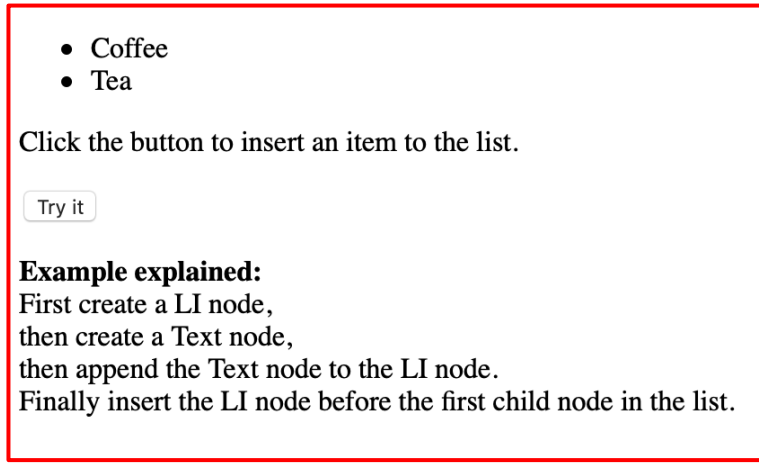
```
<!DOCTYPE html>
<html>
<head>
    <title id="myTitle">DOM Tutorial</title>
</head>
<body>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  var x = document.getElementById("myTitle").firstChild;
  var txt = "";
  txt += "Old title: " + x.nodeValue + " ";
  txt += "New title: DOM Hacking";
  document.getElementById("myTitle").firstChild.nodeValue = txt;
}
</script>

</body>
</html>
```

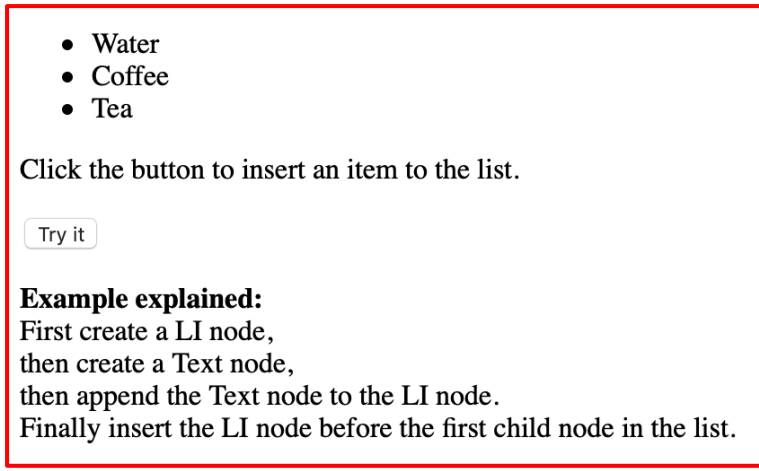
```
▼ <html>
  ▼ <head>
      <title id="myTitle">DOM Tutorial</title>
    </head>
```

```
▼ <html>
  ▼ <head>
      <title id="myTitle">Old title: DOM Tutorial New title: DOM Hacking</title>
    </head>
```

Creating new HTML elements - insertBefore

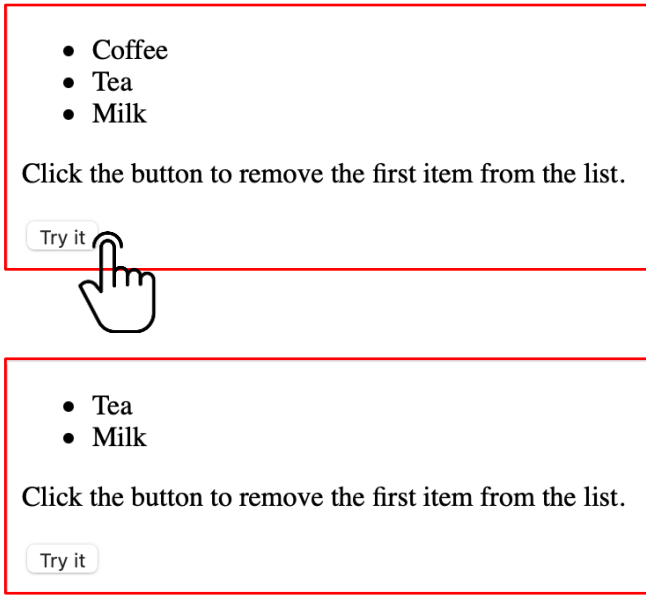
```
<!DOCTYPE html>
<html>
    <body>

        <ul id="myList">
            <li>Coffee</li>
            <li>Tea</li>
        </ul>

        <p>Click the button to insert an item to the list.</p>

        <button onclick="myFunction()">Try it</button>

        <p><strong>Example explained:</strong><br>
            First create a LI node,<br>
            then create a Text node,<br>
            then append the Text node to the LI node.<br>
            Finally insert the LI node before the first
            child node in the list.</p>

        <script>
            function myFunction() {
                var newItem = document.createElement("LI");
                var textnode = document.createTextNode("Water");
                newItem.appendChild(textnode);

                var list = document.getElementById("myList");
                list.insertBefore(newItem, list.childNodes[0]);
            }
        </script>

    </body>
</html>
```





Example: removing existing HTML elements

```
<!DOCTYPE html>
<html>
    <body>

    <!-- Note that the <li> elements inside <ul> are not indented (whitespaces).
        If they were, the first child node of <ul> would be a text node
    -->
    <ul id="myList"><li>Coffee</li><li>Tea</li><li>Milk</li></ul>

    <p>Click the button to remove the first item from the list.</p>

    <button onclick="myFunction()">Try it</button>

    <script>
        function myFunction() {
            var list = document.getElementById("myList");
            list.removeChild(list.childNodes[0]);
        }
    </script>

    </body>
</html>
```



Example: change HTML attributes

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Can Change Images</h1>

<img id="myImage" onclick="changeImage()" src="pic_bulboff.gif"
width="100" height="180">

<p>Click the light bulb to turn on/off the light.</p>

<script>
function changeImage() {
    var image = document.getElementById('myImage');
    if (image.src.match("bulbon")) {
        image.src = "pic_bulboff.gif";
    } else {
        image.src = "pic_bulbon.gif";
    }
}
</script>

</body>
</html>
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_lightbulb

Example: addEventListener method

```
<!DOCTYPE html>
<html>
<body>

<h1>The Element Object</h1>
<h2>The addEventListener() Method</h2>

<p>Attach a click event to a button:</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
const element = document.getElementById("myBtn");
element.addEventListener("click", function() {
  document.getElementById("demo").innerHTML = "Hello World";
});
</script>

</body>
</html>
```

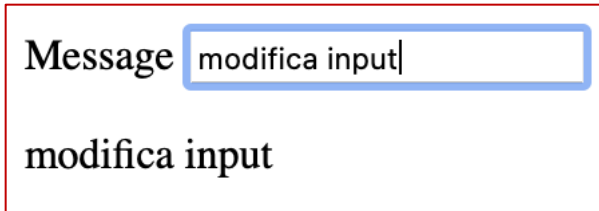
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_element_addeventlistener

Example: addEventListener method (2)

- The **input** event occurs when the value of an <input> or <textarea> element is changed
  - The **input** event does NOT occur when a <select> element changes
- The **input** event is similar to the **change** event
  - The difference is that the **input** event occurs immediately after the content has been changed, while change occurs when the element loses focus

```
<label for="message">Message</label>
<input placeholder="Enter some text" id="message" name="message">
<p id="result"></p>
<script>
    const message = document.getElementById("message");
    const result = document.getElementById("result");

    message.addEventListener('input', function () {
        result.innerHTML = this.value;
    });
</script>
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oninput



Form

- Un documento può contenere più oggetti form
- Un oggetto form può essere referenziato con il suo nome (attributo **name**) o mediante il vettore **forms[]** esposto da **document**:

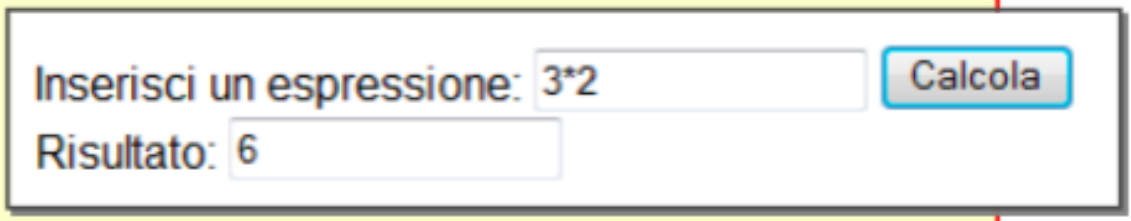
  document.*nomeForm*
  document.forms[n]
  document.forms["*nomeForm*"]
  document.forms["*idForm*"]
- Gli elementi del form possono essere referenziati con il loro nome o mediante il vettore **elements[]**

  document.*nomeForm*.*nomeElemento*
  document.forms[n].elements[m]
  document.forms["*nomeForm*"].elements[*"nomeElemento"*]
  document.forms["*idForm*"].elements[*"idElemento"*]
- Ogni elemento ha una proprietà **form** che permette di accedere al form che lo contiene (vedi l'esempio "calcolatrice" che segue)

Esempio: calcolatrice (calcolatrice.html)

```
<head>
  <script type="text/javascript">
     function compute(f)
     {
       if (confirm("Sei sicuro?"))
         f.result.value = eval(f.expr.value);
       else alert("Ok come non detto");
     }
  </script>
</head>
<body>
  <form>
    Inserisci un'espressione:
    <input type="text" name="expr" size=15 >
    <input type="button" value="Calcola"
       onClick="compute(this.form)"><br/>
    Risultato:
    <input type="text" name="result" size="15" >
  </form>
</body>
```

eval: evaluate/execute JavaScript code/expressions



riferimento al form che contiene il controllo

Form (2)

- Proprietà delle form:
  - **action**: riflette l'attributo action
  - **elements**: vettore contenente gli elementi della form (https://www.w3schools.com/html/html_form_elements.asp)
  - **length**: numero di elementi nella form
  - **method**: riflette l'attributo method
  - **name**: nome del form
  - **…**
- Metodi delle form:
  - **reset()**: resetta il form
  - **submit()**: esegue il submit
- Eventi delle form:
  - **reset**: quando il form viene resettato
  - **submit**: quando viene eseguito il submit del form

Example: access to the form elements (accessformelement.html)

```
<!DOCTYPE html>
<html>
    <body>

        <h3>A demonstration of how to access a FORM element</h3>

        <form id="myForm" action="/action_page.php">
            First name: <input type="text" name="fname" value="Donald"><br>
            Last name: <input type="text" name="lname" value="Duck"><br>
            <input type="submit" value="Submit">
        </form>

        <p>Click the "Try it" button to get the URL for where to send the form data when
           the form above is submitted.</p>

        <button onclick="myFunction()">Try it</button>

        <p id="demo"></p>

        <script>
            function myFunction() {
                var x = document.getElementById("myForm").action;
                document.getElementById("demo").innerHTML = x;

                myForm.fname.value = "Try Donald";
                myForm.lname.value = "Try Duck";
            }
        </script>

    </body>
</html>
```

I controlli di un form

- Ogni tipo di controllo (widget) che può entrare a far parte di un form è rappresentato da un oggetto JavaScript:
  - **Text: <input type =“text”>**
  - **Checkbox: <input type=“checkbox”>**
  - **Button: <input type=“button”> o <button>**
  - **Radio: <input type=“radio”>**
  - **Hidden: <input type=“hidden”>**
  - **File: <input type=“file”>**
  - **Password: <input type=“password”>**
  - **Textarea: <textarea>**
  - **Submit: <input type=“submit”>**
  - **Reset: <input type=“reset”>**
  - **…**

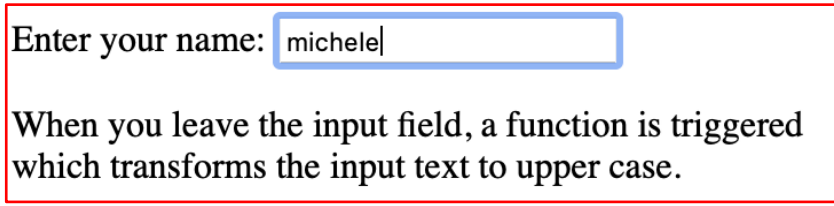
```
var checkbox = document.createElement('input');
checkbox.type = "checkbox";
checkbox.name = "name";
checkbox.value = "value";
checkbox.id = "id";
//form container (reference to a form node)
container.appendChild(checkbox);
var label = document.createElement('label')
label.htmlFor = "id";
label.appendChild(
            document.createTextNode(’text….’));
container.appendChild(label);
```

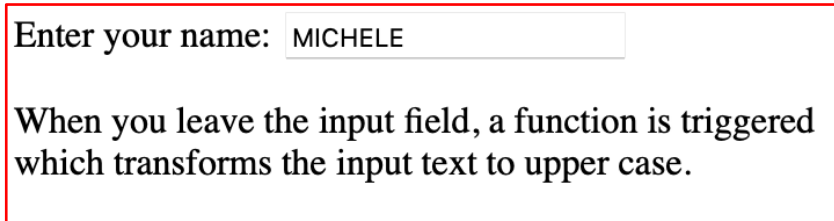
Proprietà, metodi ed eventi comuni ai vari controlli delle form

- Proprietà (get/set):
  - **form**: riferimento al form che contiene il controllo
  - **name**: nome del controllo
  - **type**: tipo del controllo
  - **value**: valore dell'attributo value
  - **disabled**: disabilitazione/abilitazione del controllo
- Metodi:
  - **blur()** toglie il focus al controllo
  - **focus()** dà il focus al controllo
  - **click()** simula il click del mouse sul controllo
- Eventi:
  - **blur** quando il controllo perde il focus
  - **focus** quando il controllo prende il focus
  - **click** quando l'utente clicca sul controllo
  - **change** quando il controllo cambia e perde il focus

Example: blur event

```
<!DOCTYPE html>
<html>
    <body>

      Enter your name: <input type="text" id="fname" onblur="myFunction()">

      <p>When you leave the input field, a function is triggered which transforms
         the input text to upper case.</p>

      <script>
          function myFunction() {
              var x = document.getElementById("fname");
              x.value = x.value.toUpperCase();
          }
      </script>

    </body>
</html>
```



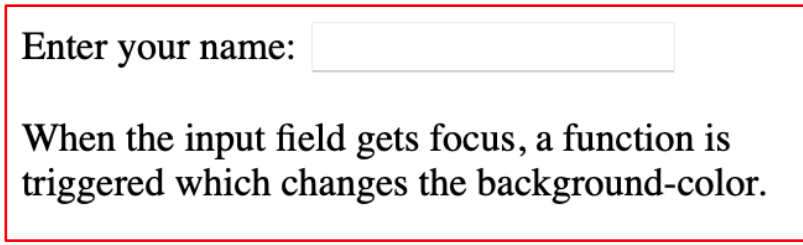


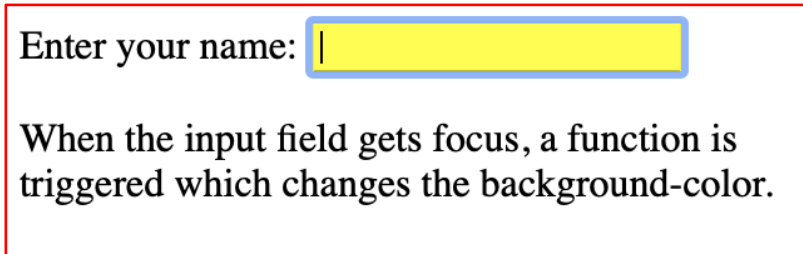
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onblur

Example: focus event

```
<!DOCTYPE html>
<html>
    <body>

      Enter your name: <input type="text" onfocus="myFunction(this)">

      <p>When the input field gets focus, a function is triggered which
         changes the background-color.</p>

      <script>
          function myFunction(x) {
              x.style.background = "yellow";
          }
      </script>

    </body>
</html>
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onfocus

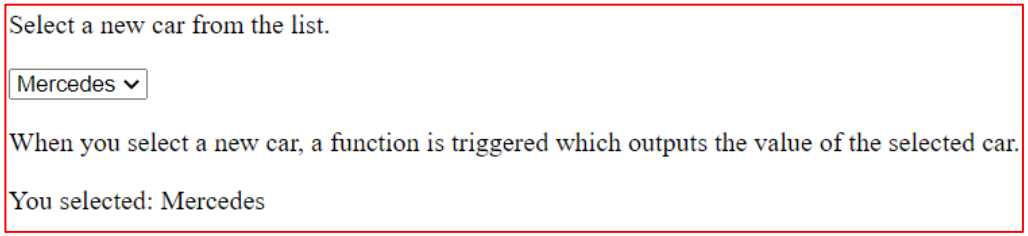




Example: change event

- This event is similar to the **input** event.
  - The difference is that the input event occurs immediately after the value of an element has changed, while change occurs when the element loses focus, after the content has been changed
  - The other difference is that the change event also works on <select> elements

```
<body>

<p>Select a new car from the list.</p>

<select id="mySelect" onchange="myFunction()">
  <option value="Audi">Audi</option>
  <option value="BMW">BMW</option>
  <option value="Mercedes">Mercedes</option>
  <option value="Volvo">Volvo</option>
</select>

<p>When you select a new car, a function is triggered which outputs the value of the
selected car.</p>

<p id="demo"></p>

<script>
function myFunction() {
  var x = document.getElementById("mySelect").value;
  document.getElementById("demo").innerHTML = "You selected: " + x;
}
</script>

</body>
```



https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onchange

Proprietà e metodi comuni ai controlli text e password

- Proprietà (get/set):
  - **defaultValue** valore di default
  - **maxLength** numero massimo di caratteri
  - **readOnly** sola lettura / lettura e scrittura
  - **size** dimensione del controllo
- Metodi:
  - **select()** seleziona il testo
    - https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_password_select

Proprietà comuni a checkbox e radio

- Proprietà (get/set):
  - **checked**: dice se il box è spuntato o il radio button è selezionato
  - **defaultChecked**: impostazione di default

Validazione di un form

- Uno degli utilizzi più frequenti di JavaScript è nell'ambito della validazione dei campi di un form
  - Riduce il carico delle applicazioni server side filtrando l'input
  - Riduce il ritardo in caso di errori di inserimento dell'utente
  - Consente di introdurre dinamicità all'interfaccia web
- Generalmente si valida un form in due momenti:
  1. Durante l'inserimento utilizzando l'evento **onChange()** sui vari controlli
  2. Al momento del submit utilizzando l'evento **onClick()** del bottone di submit o l'evento **onSubmit()** del form

Esempio di validazione (submit event)

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>

<h2>JavaScript Validation</h2>

<form name="myForm" action="/action_page.php" onsubmit="return validateForm()"
method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

La funzione di validazione deve restituire true, se la validazione è avvenuta con successo, false altrimenti

- Restituendo false, si impedisce il submit della form

Non dimenticare il return nel codice dell'event Handler

https://www.w3schools.com/js/tryit.asp?filename=tryjs_validation_js

Esempio di validazione (click event)

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>

<h2>JavaScript Validation</h2>

<form name="myForm" action="/action_page.php" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit" onclick="return validateForm()">
</form>

</body>
</html>
```

The constraint validation API

- It consists of a set of methods and properties for form elements, which are used for constraint validation
- **Constraint Validation Methods**

Method	Description
checkValidity()	Returns true if an input element contains valid data.
setCustomValidity()	Sets the validationMessage property of an input element.

- **Constraint Validation Fields**

Property	Description
validity	Contains boolean properties related to the validity of an input element.
validationMessage	Contains the message a browser will display when the validity is false.
willValidate	Indicates if an input element will be validated.

The constraint validation API (2)

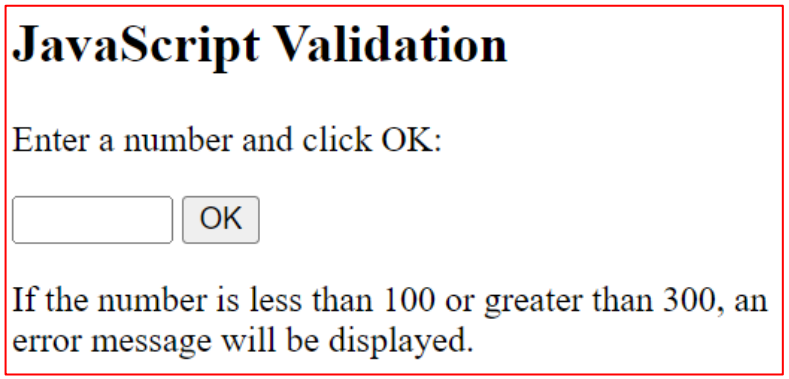
- The **validity** property of an input element contains a number of properties related to the validity of data:

Property	Description
customError	Set to true, if a custom validity message is set.
patternMismatch	Set to true, if an element's value does not match its pattern attribute.
rangeOverflow	Set to true, if an element's value is greater than its max attribute.
rangeUnderflow	Set to true, if an element's value is less than its min attribute.
stepMismatch	Set to true, if an element's value is invalid per its step attribute.
tooLong	Set to true, if an element's value exceeds its maxLength attribute.
typeMismatch	Set to true, if an element's value is invalid per its type attribute.
valueMissing	Set to true, if an element (with a required attribute) has no value.
valid	Set to true, if an element's value is valid.

The constraint validation API (3)

```
<h2>JavaScript Validation</h2>

<p>Enter a number and click OK:</p>

<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">OK</button>

<p>If the number is less than 100 or greater than 300, an error message will
be displayed.</p>

<p id="demo"></p>

<script>
function myFunction() {
  const inpObj = document.getElementById("id1");
  if (!inpObj.checkValidity()) {
    document.getElementById("demo").innerHTML = inpObj.validationMessage;
  } else {
    document.getElementById("demo").innerHTML = "Input OK";
  }
}
</script>
```

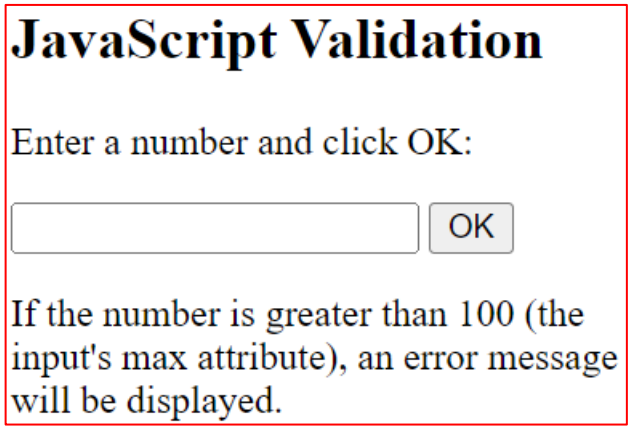
https://www.w3schools.com/js/tryit.asp?filename=tryjs_validation_check



The constraint validation API (4)

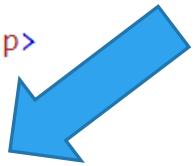
```
<h2>JavaScript Validation</h2>

<p>Enter a number and click OK:</p>

<input id="id1" type="number" max="100">
<button onclick="myFunction()">OK</button>

<p>If the number is greater than 100 (the input's max attribute), an error
message will be displayed.</p>

<p id="demo"></p>

<script>
function myFunction() {
  let text;
  if (document.getElementById("id1").validity.rangeOverflow) {
    text = "Value too large";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_validation_rangeOverflow



JavaScript errors - throw and try…catch

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Errors</h1>
<p>In this example we have written adddlert to deliberately produce an error:</p>

<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>

</body>
</html>
```



JavaScript Errors

In this example we have written adddlert to deliberately produce an error:

adddlert is not defined

https://www.w3schools.com/js/tryit.asp?filename=tryjs_try_catch

Example: finally statement

```
<!DOCTYPE html>
<html>
<body>

<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test
Input</button>

<p id="message"></p>

<script>
function myFunction() {
    var message, x;
    message = document.getElementById("message");
    message.innerHTML = "";
    x = document.getElementById("demo").value;
    try {
        if(x == "")  throw "is empty";
        if(isNaN(x)) throw "is not a number";
        x = Number(x);
        if(x > 10)   throw "is too high";
        if(x < 5)    throw "is too low";
    }
    catch(err) {
        message.innerHTML = "Input " + err;
    }
    finally {
        document.getElementById("demo").value = "";
    }
}
</script>
</body>
</html>
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_throw_error

The debugger keyword

- The **debugger** keyword stops the execution of JavaScript, and calls (if available) the debugging function
- This has the same function as setting a breakpoint in the debugger
- If no debugging is available, the debugger statement has no effect

```
var x = 15 * 5;
debugger;
document.getElementbyId("demo").innerHTML = x;
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_state_debugger

Riferimenti

- Tutorial (JavaScript, HTML DOM):
  - **http://www.w3schools.com/js/default.asp**