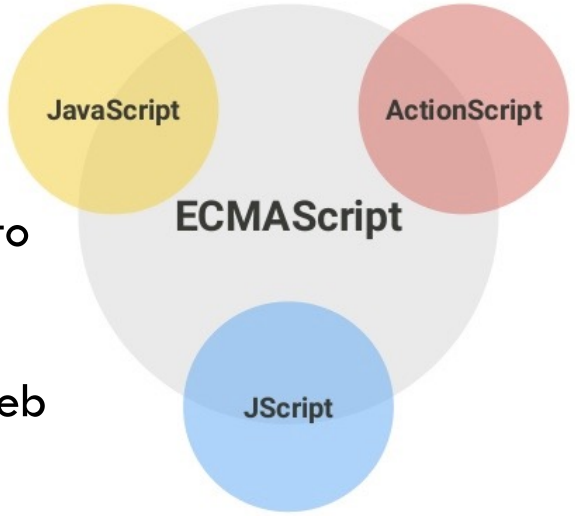
CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

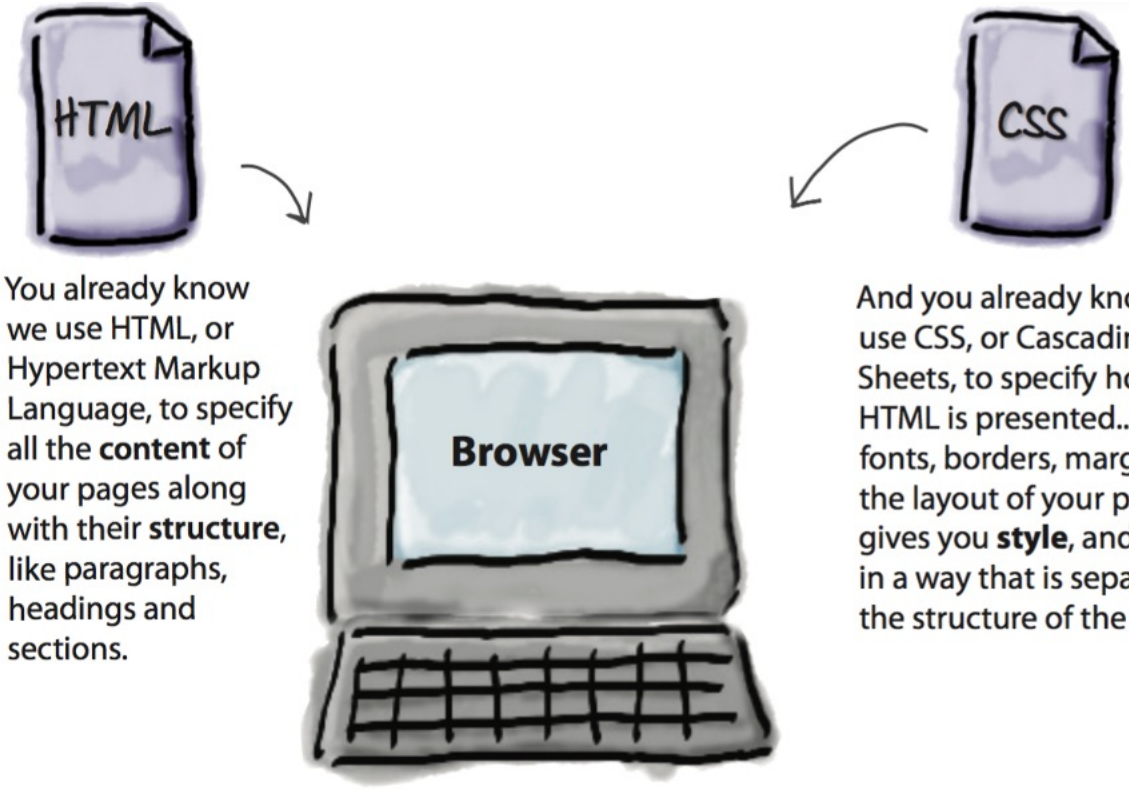
JAVASCRIPT

a.a. 2020-2021

Che cos'è JavaScript



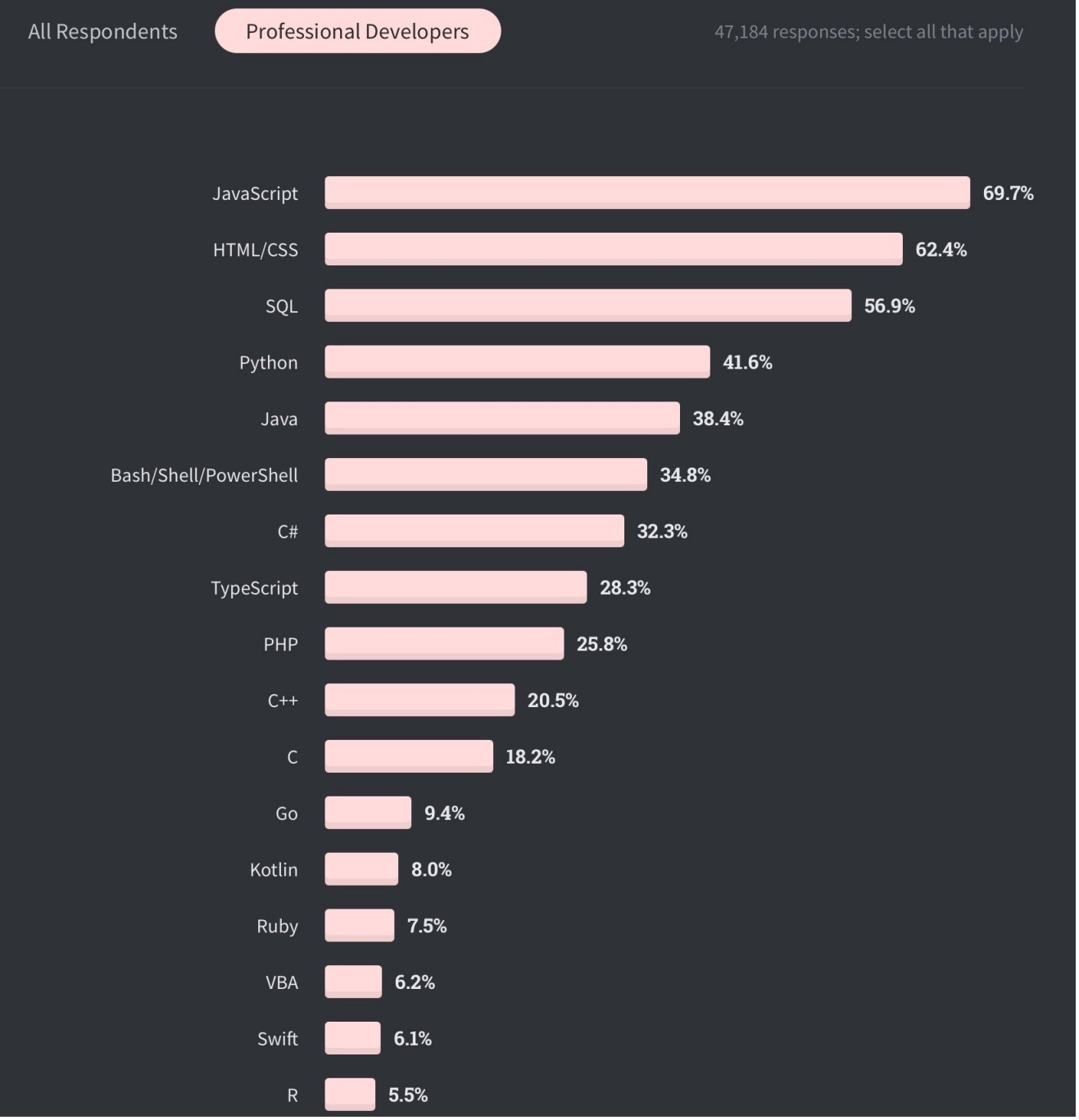
- JavaScript è un linguaggio di **scripting** sviluppato per dare interattività alle pagine HTML
- Può essere inserito direttamente nelle pagine Web ed è in pratica lo standard client-side
- Il suo nome ufficiale è ECMAScript
  - È diventato standard ECMA (European Computer Manufactures Association) (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.

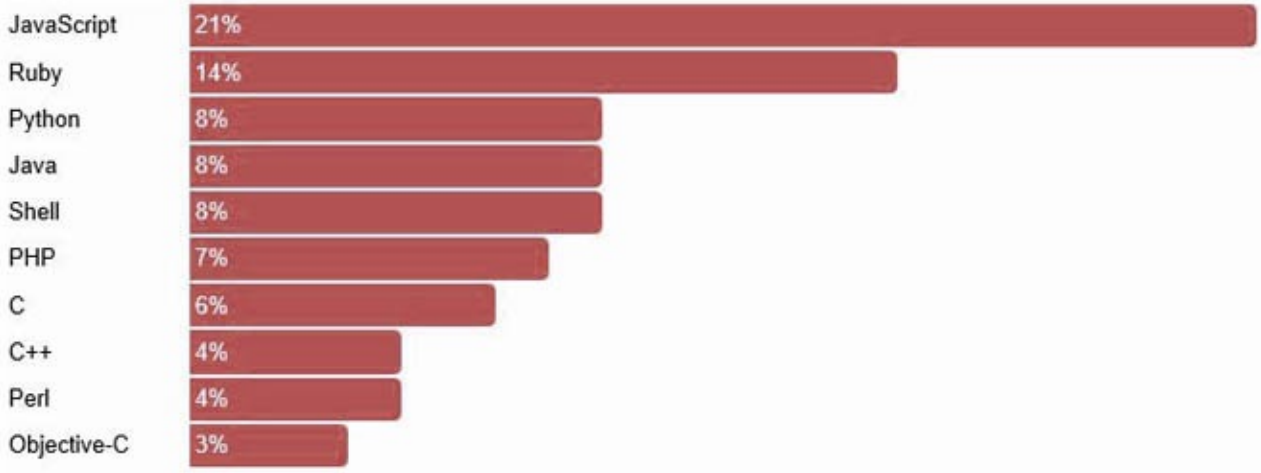


Most popular technologies

https://insights.stackoverflow.com/survey/2020

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)



JavaScript (ECMAScript 5) 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 **object-based** ma **non class-based** (introdotte nell'ECMAScript 6)
    - Esiste il concetto di oggetto
    - Non esiste il concetto di classe
  - JavaScript è **debolmente tipizzato** *(weakly typed)*
    - Non è necessario definire il tipo di una variabile
    - *Attenzione: questo non vuol dire che i dati non abbiano un tipo (sono le variabili a non averlo in modo statico)*

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

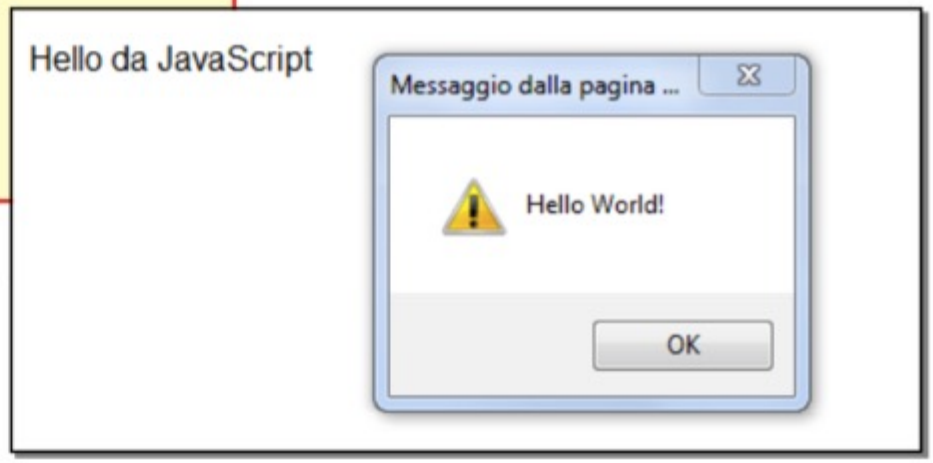
Cosa si può fare con JavaScript

- 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.

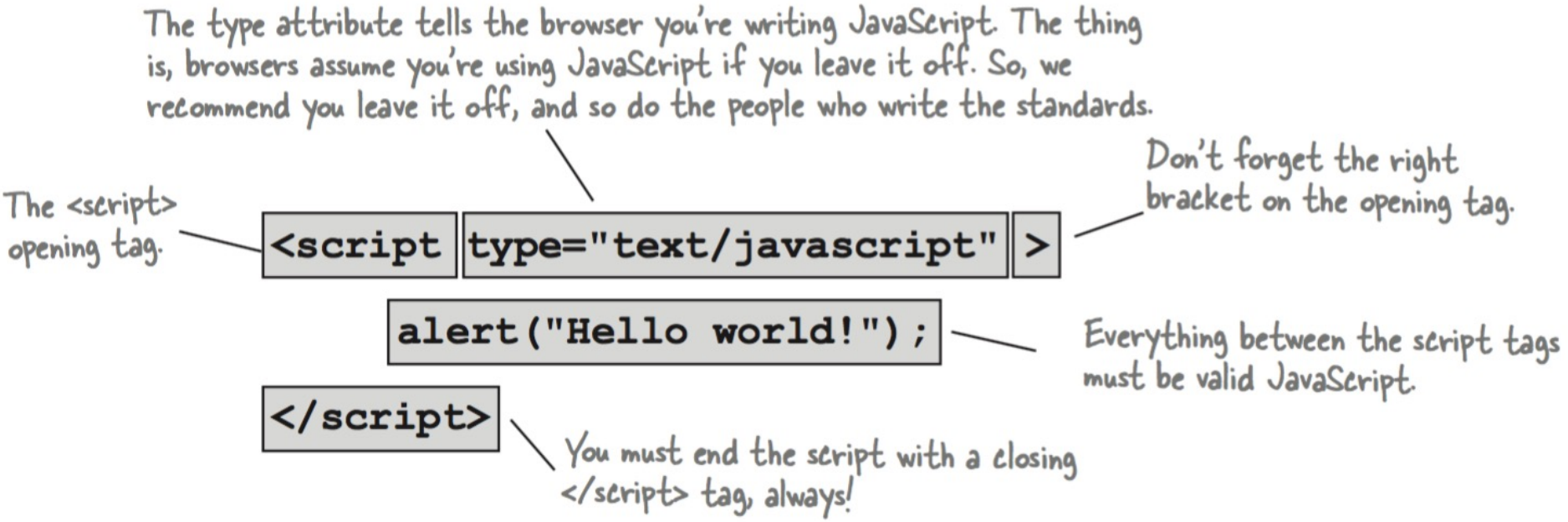
Esempio

- 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>
```



Script element



Example: setTimeout anf function (test.html)

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>
```

alert writes into an alert box (**window.alert**)

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

Variabili

- Le variabili vengono dichiarate usando la parola chiave **var**

  var nomevariabile;

- **Non hanno un tipo**
  - possono contenere valori di qualunque tipo
- Si può inizializzare contestualmente alla dichiarazione:

  var f = 15.8;

- Possono essere dichiarate in linea:

  for (var i = 1, i<10, i++) { … }

- Esiste lo **scope globale**, **locale** (ovvero dentro una funzione) e lo **scope di blocco** definito utilizzando la keyword **let** (ECMAScript 6)
- Una variabile **const** (ECMAScript 6) assegna un valore che non può essere cambiato

Example

```
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
```

JavaScript keywords

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 or two options

- Single option

```
if (condition) {
  statement_1;
  ...
  statement_N;
}
```

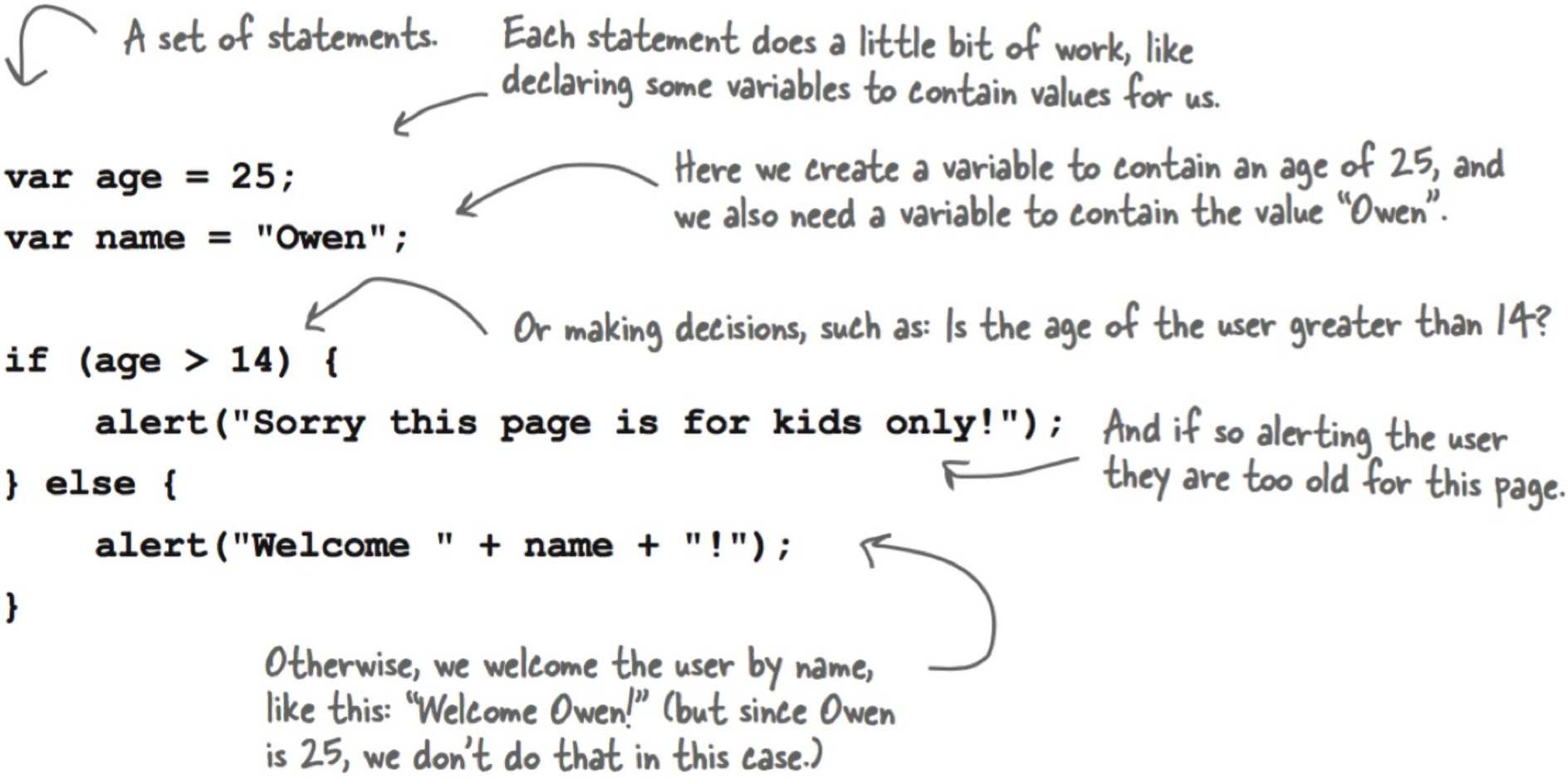
- Two options

```
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 (including the string "false")

How to make a statement



Decisions

```
if (scoops >= 5) {
    alert("Eat faster, the ice cream is going to melt!");
} else if (scoops < 3) {
    alert("Ice cream is running low!");
}
```

another test with if/else if

Add as many tests with "else if" as you need, each with its own associated code block that will be executed when the condition is true.

Example: alert and decisions (icecream.html)

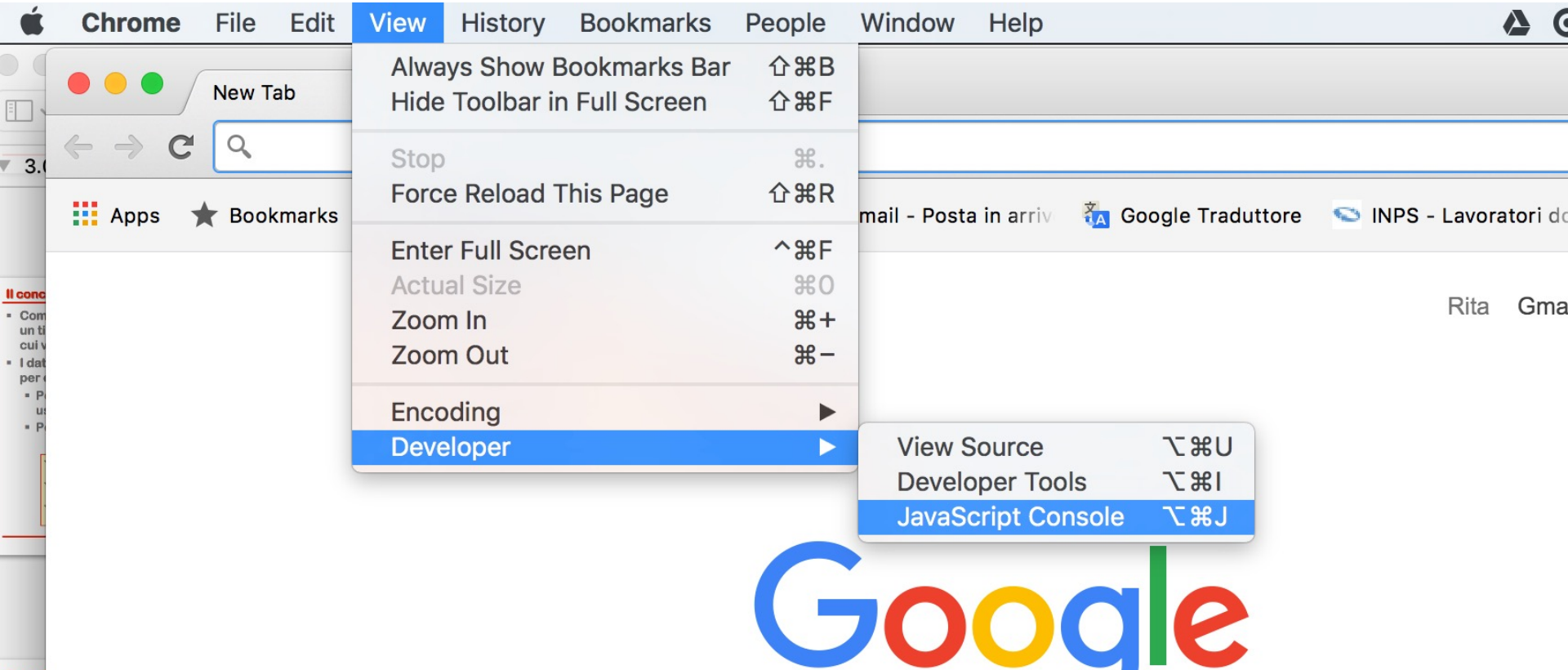
```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Icecream!</title>
  </head>
  <body>
  </body>
  <script>
    scoops = 5;
    while (scoops > 0) {
        document.write("Another scoop!<br>");
        if (scoops < 3) {
            alert("Ice cream is running low!");
        } else if (scoops >= 5) {
            alert("Eat faster, the ice cream is going to melt!");
        }
        scoops = scoops - 1;
    }
    document.write("Life without ice cream isn't the same");
  </script>
</body>
</html>
```

document.write writes some text directly to the HTML document

Using document.write() after an HTML document is loaded, will delete all existing HTML

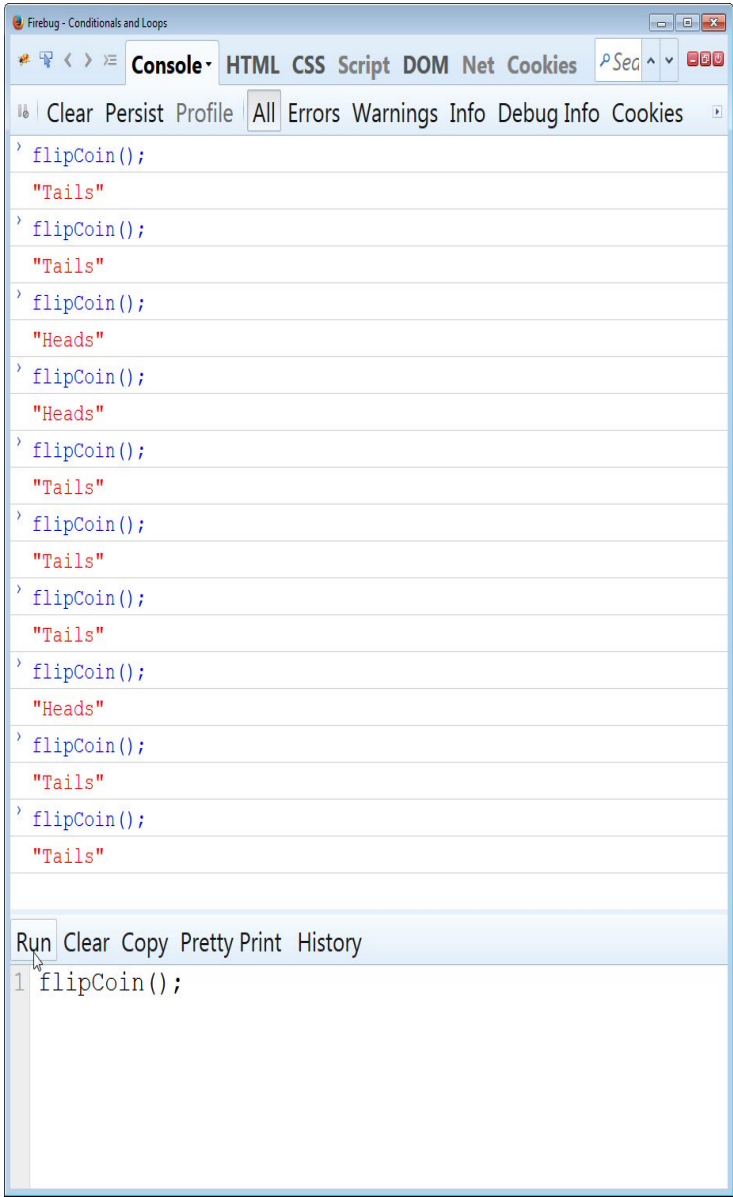
Opening the console

- 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

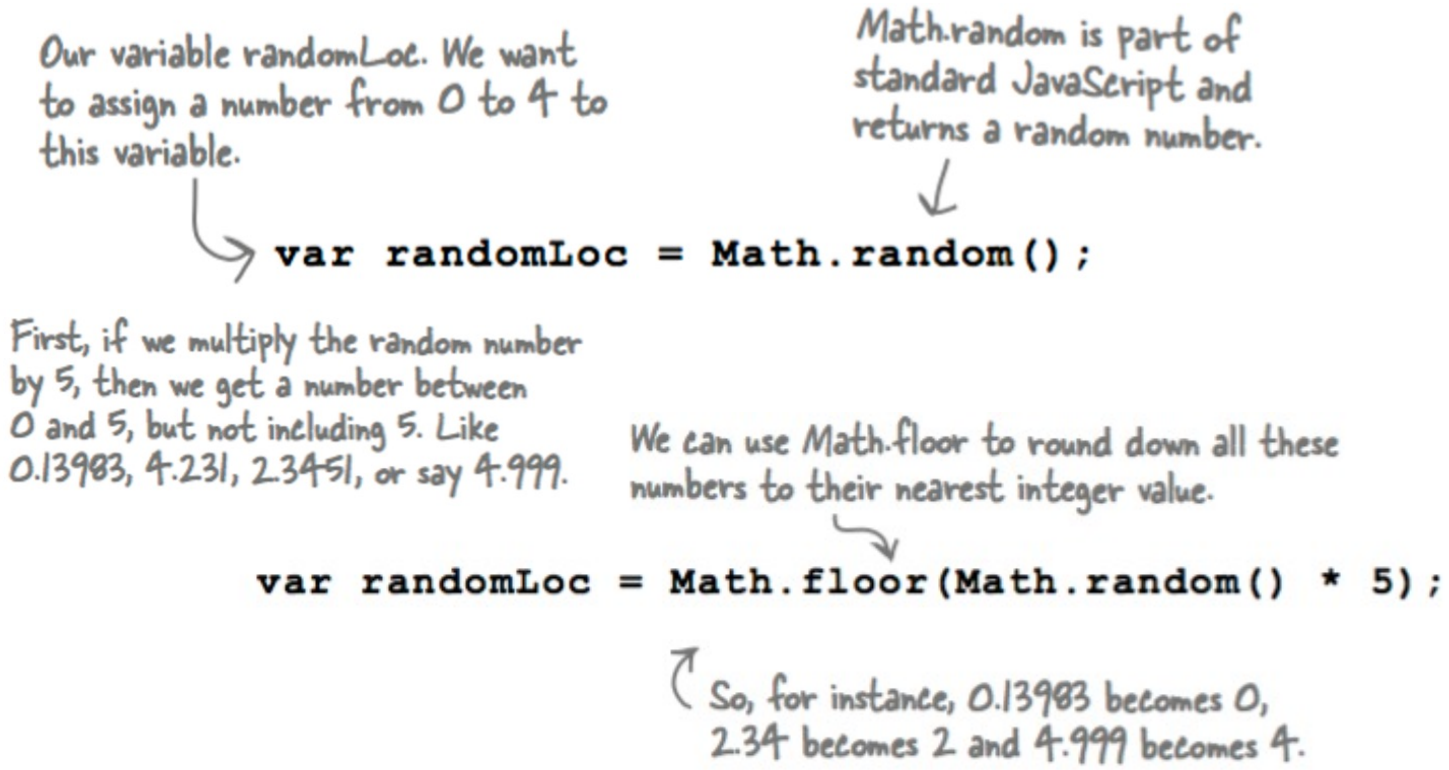
```
function flipcoin() {
  if (Math.random() < 0.5) {
    return("heads");
  } else {
    return("tails");
  }
}
alert(flipcoin());
```



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

Random

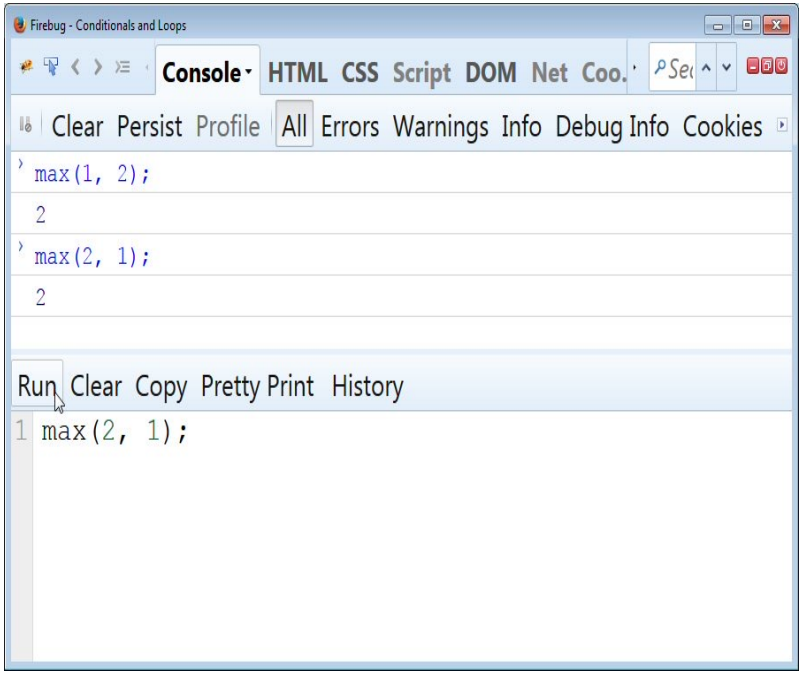
- We need is an integer between 0 and 4



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

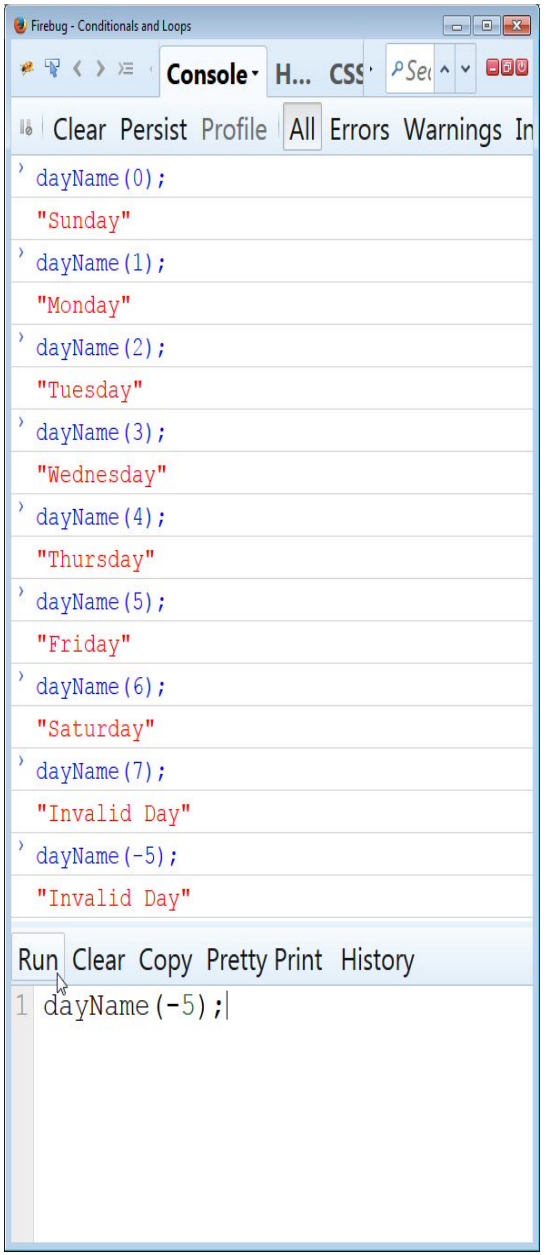
Example: if-else

```
function max(n1, n2) {
  if (n1 >= n2) {
    return(n1);
  } else {
    return(n2);
  }
}
```



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);
}
```



Boolean operators

- **==, !=**
  - Equality, inequality
- **===**
  - In addition to comparing primitive types, === tests if two objects are identical (the same object), not just if they appear equal (have the same fields). More details when we introduce objects. Not used frequently
- **<, <=, >, >=**
  - Numeric less than, less than or equal to, greater than, greater than or equal to
- **&&, ||**
  - Logical and, or. Both use ***short-circuit evaluation*** to more efficiently compute the results of complicated expressions
- **!**
  - Logical negation

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

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

JavaScript 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

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
<<=	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
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

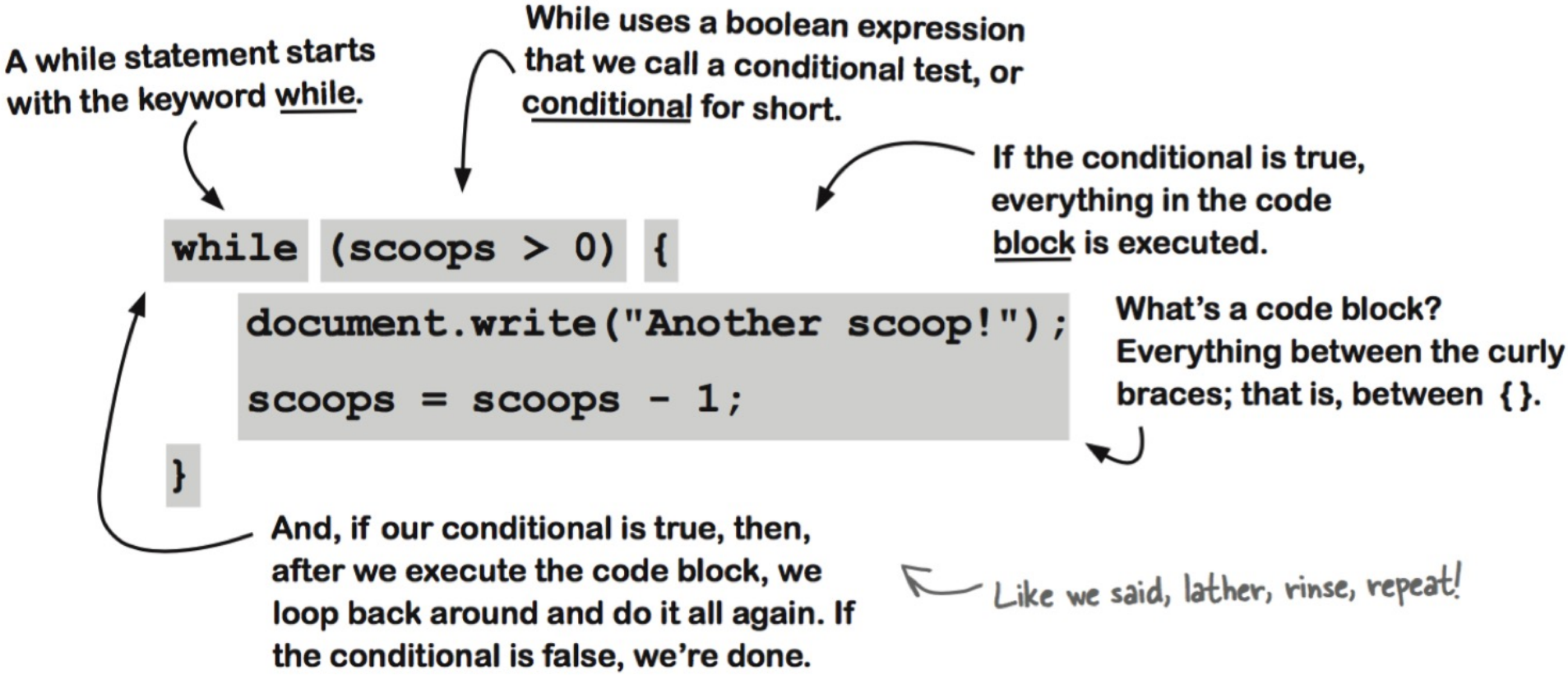
```
txt1 = "What a very ";
txt1 += "nice day";
```

JavaScript bitwise operators

- Bit operators work on 32 bits numbers

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

While statement



Example: Happy birthday (birthday.html)

```
<!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>
```

Example: console output (bottle.html)

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>My First JavaScript</title>
</head>
<body>
<script>
var word = "bottles";
var count = 99;
while (count > 0) {
    console.log(count + " " + word + " of beer on the wall");
    console.log(count + " " + word + " of beer,");
    console.log("Take one down, pass it around,");
    count = count - 1;
    if (count > 0) {
        console.log(count + " " + word + " of beer on the wall.");
    } else {
        console.log("No more " + word + " of beer on the wall.");
    }
}
</script>
</body>
</html>
```

console.log() method 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: numeri e booleani

- JavaScript prevede pochi tipi primitivi: **numeri**, **booleani** e **stringhe**
- 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
- I dati hanno un tipo e per ogni tipo esiste una sintassi per esprimere le costanti (**literal**)
- Per i numeri, ad esempio, le costanti hanno la forma usuale: **1.0**, **3.5** o in altre basi (i.e., 015, 0x123A, 0b110)
- Per i booleani sono gli usuali valori **true** e **false**

```
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 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
- **You can link to an external file in the body of your page**
  - The best of both worlds. 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

How do I add code to my page? (3)

- Open “bottle.html” and select all the code; that is, everything between the <script> tags
- Now create a new file named “code.js” in your editor, and place the code into it. Then save “code.js”

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script src="code.js">

    </script>
  </body>
</html>
```

Use the src attribute of the <script> element to link to your JavaScript file.

Where your code was.

Believe it or not we still need the ending <script> tag, even if there is no code between the two tags.

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>
  - Some prefer the equivalent version prefaced with "./"

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

Loading scripts: alternative approach

- Script tag with body content (in body section of HTML page)

```
<body>
<script>JavaScript code that uses document.write(…)
</script>
</body>
```

- Purpose
  - To directly invoke code that will run as page loads
    - e.g., to output HTML content built by JavaScript using **document.write**
  - Don't use this approach for defining functions
    - Slower (no browser caching) and less reusable

Example: code.js

```
function getmessage() {
  var amount = Math.round(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

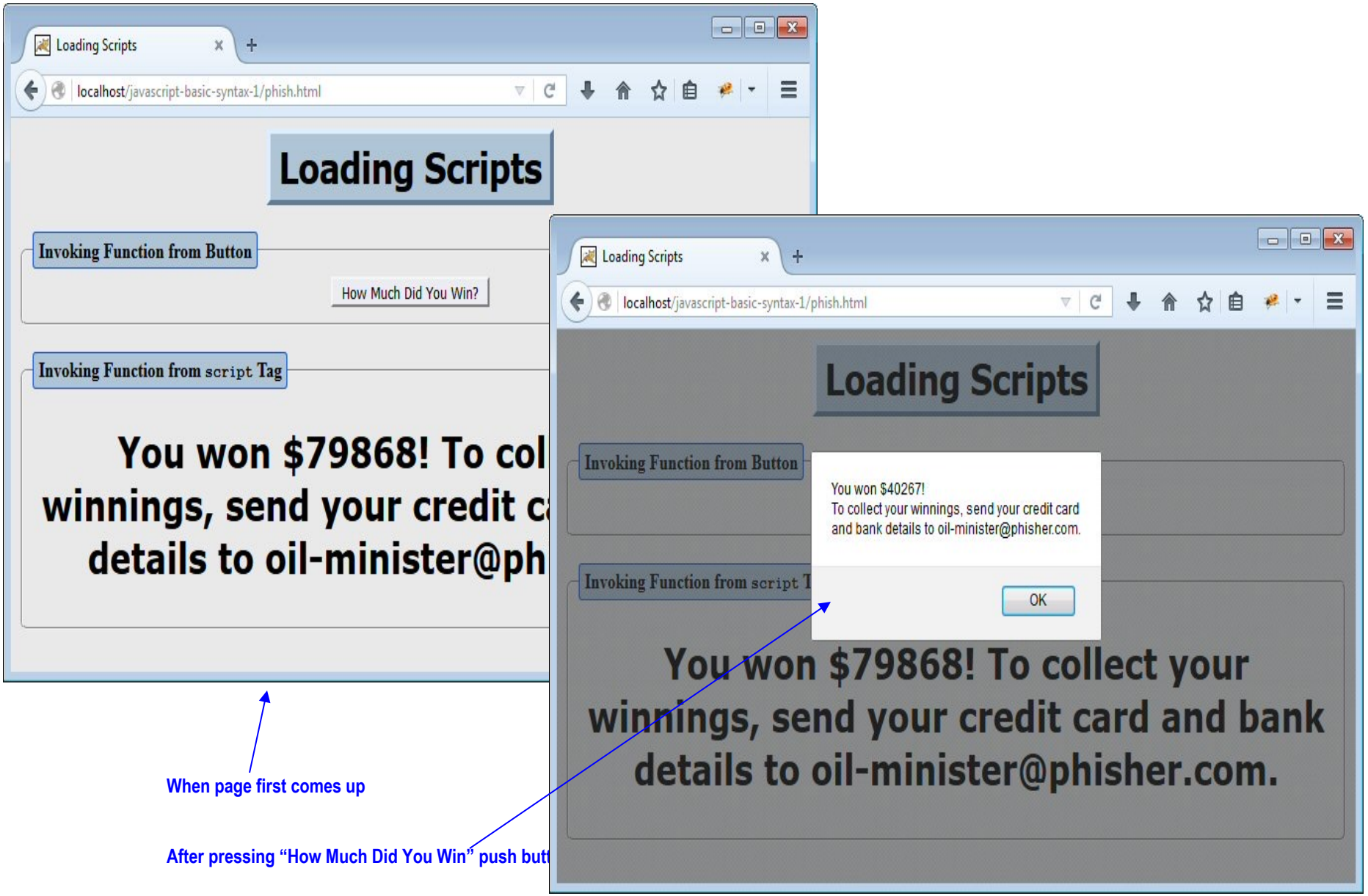
"document.write" inserts text into page at location that it is called

Example: import code.js in an html document

```
<!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

Example (Results)

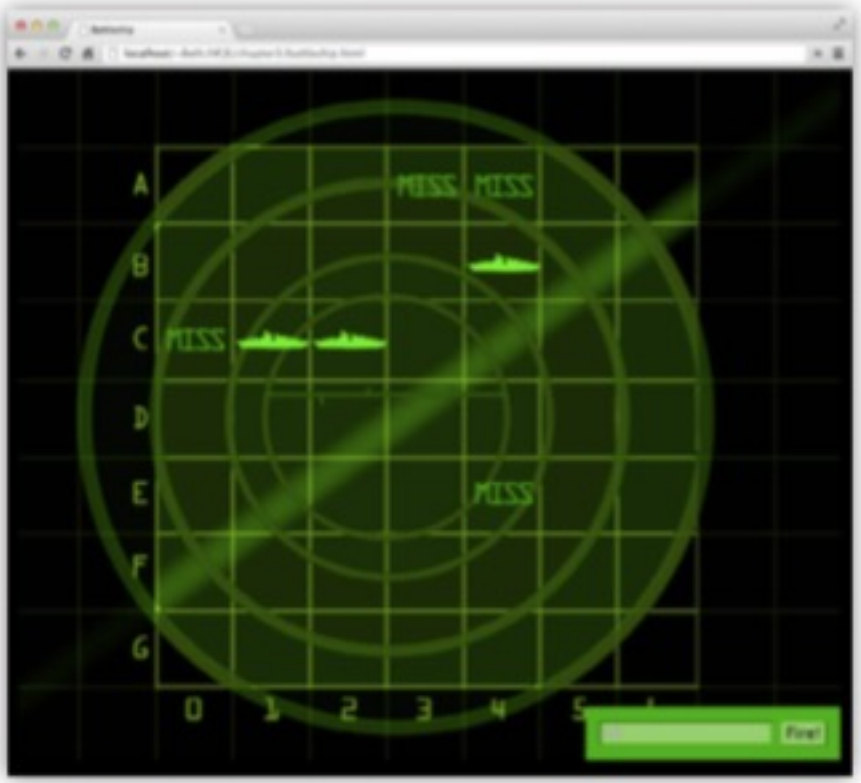


Exercise: battleship game

- **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

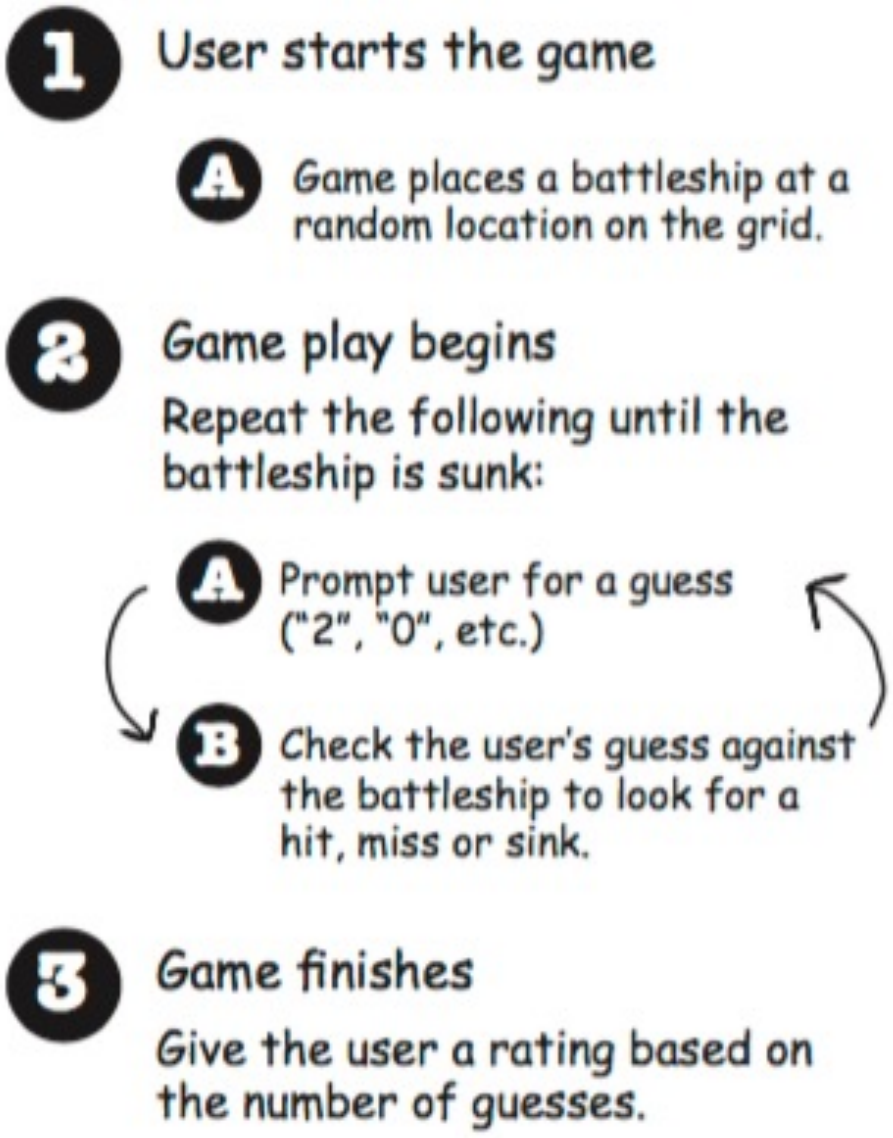
- *Objective: 7x7 graphical version with three ships, but…*
- We're going to start with a nice 1-D grid with seven locations and one ship to find



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)

DECLARE three *variables* to hold the location of each cell of the ship. Let's call them location1, location2 and location3
DECLARE a *variable* to hold the user's current guess. Let's call it guess
DECLARE a *variable* to hold the number of hits. We'll call it hits and *set* it to 0
DECLARE a *variable* to hold the number of guesses. We'll call it guesses and *set* it to 0
DECLARE a *variable* to keep track of whether the ship is sunk or not. Let's call it isSunk and *set* it to false

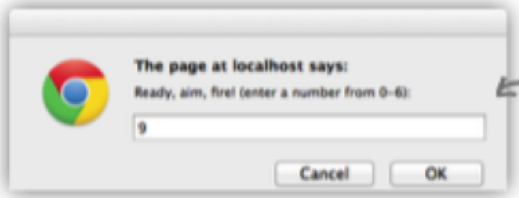
```
var randomLoc = Math.floor(Math.random() * 5);
var location1 = randomLoc;
var location2 = location1 + 1;
var location3 = location1 + 2;
var guess;
var hits = 0;
var guesses = 0;
var isSunk = false;
```

Battleship (3) *Find the trick!*

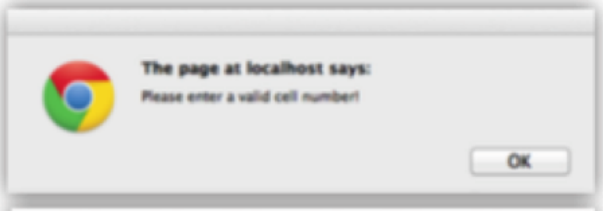
```
while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
        alert("Please enter a valid cell number!");
    } else {
        guesses = guesses + 1;
        if (guess == location1 || guess == location2 || guess == location3) {
            alert("HIT!");
            hits = hits + 1;
            if (hits == 3) {
                isSunk = true;
                alert("You sank my battleship!");
            }
        } else {
            alert("MISS");
        }
    }
}
var stats = "You took " + guesses + " guesses to sink the battleship, " +
            "which means your shooting accuracy was " + (3/guesses);
alert(stats);
```

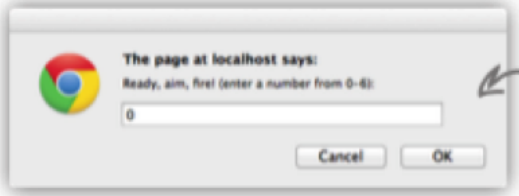
Test it!

Here's what our game interaction looked like.

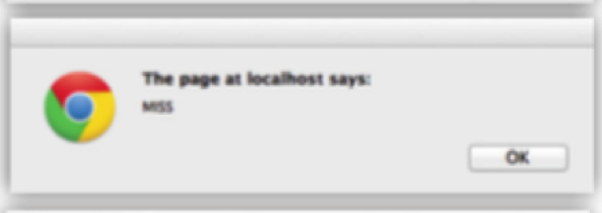


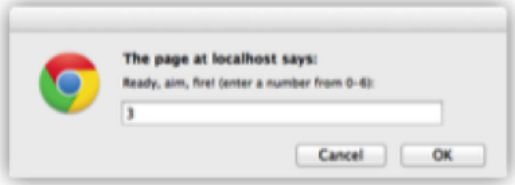
First we entered an invalid number, 9.



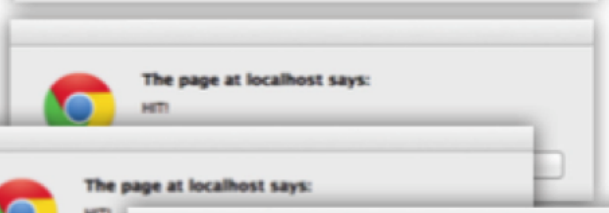


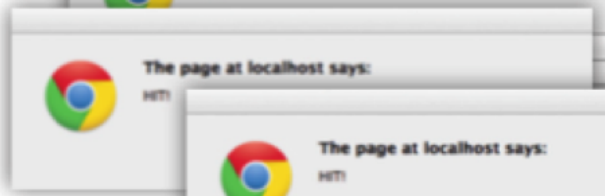
Then we entered 0, to get a miss.

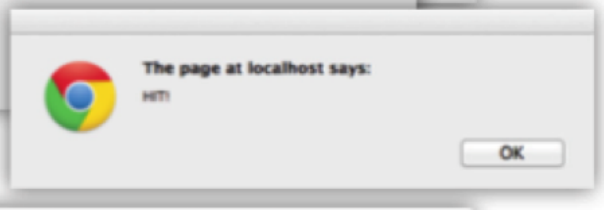


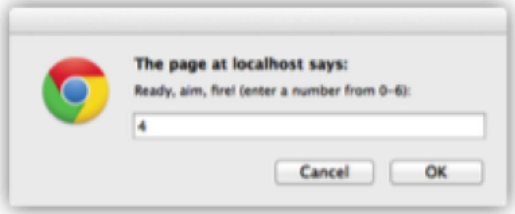


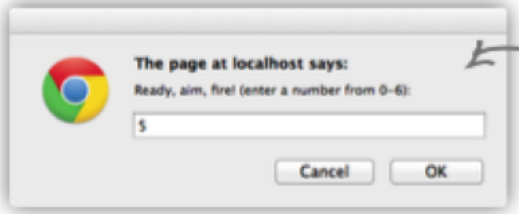
But then we get three hits in a row!



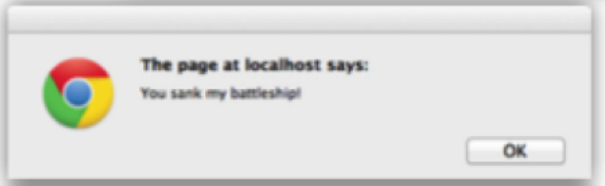




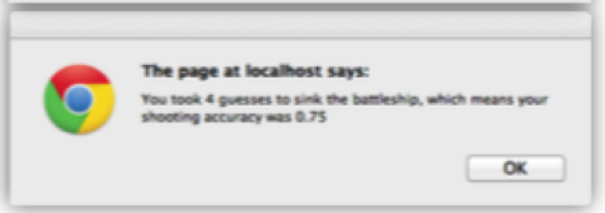




On the third and final hit, we sink the battleship.



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)
  - Le proprietà non sono definite a priori: *possono essere aggiunte dinamicamente*
- Gli oggetti vengono creati usando l'operatore **new**:

  var o = new Object()

- ***Attenzione:*** Object() è un costruttore e non una classe
- Le classi non esistono e quindi i due concetti non si sovrappongono come avviene in Java

Costruire un oggetto

- Un oggetto appena creato è completamente vuoto non ha ne proprietà ne metodi
- 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

- Le costanti oggetto (object literal) sono racchiuse fra parentesi graffe **{ }** e contengono un elenco di attributi nella forma: **nome:valore**

  var nomeoggetto = { prop1:val1, prop2:val2, ... }

- Usando le costanti oggetto creiamo un oggetto e le proprietà (valorizzate) nello stesso momento
- I due esempi seguenti sono del tutto equivalenti:

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

Sconsigliato! Non dichiarare mai oggetti Number, String, Boolean o Object

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

Don’t use new Object()

- 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()`

Example: object

```
<!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>
```

- 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
- You can use directly **demo.innerHTML**

Creating a JavaScript Objects

John is 50 years old.

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>
```

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
- Vengono istanziati con **new Array([*dimensione*])**
- Si possono creare e inizializzare usando delle costanti **array** (array literal) delimitate da [ ]:

$$\textbf{var } varname = [val_1, val_2, \ldots, val_n]$$

  - Es. **var a = [1, 2, 3];**
- Possono contenere elementi di tipo eterogeneo:
  - Es. **var b = [1, true, "ciao", {x:1,y:2}];**

Example: array (makePhrase.html)

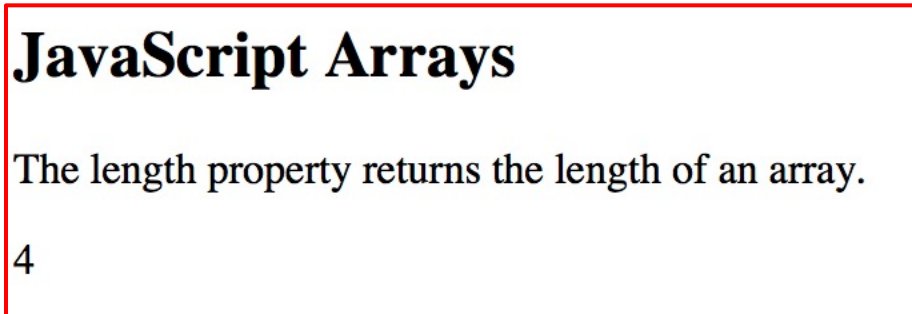
```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>makePhrase</title>
  <script>
    function makePhrases() {
      var words1 = ["24/7", "multi-tier", "30,000 foot", "B-to-B", "win-win"];
      var words2 = ["empowered", "value-added", "oriented", "focused", "aligned"];
      var words3 = ["process", "solution", "tipping-point", "strategy", "vision"];
      var rand1 = Math.floor(Math.random() * words1.length);
      var rand2 = Math.floor(Math.random() * words2.length);
      var rand3 = Math.floor(Math.random() * words3.length);
      var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
      alert(phrase);
    }
    makePhrases();
  </script>
</head>
<body></body>
</html>
```

Example: print an array

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

<h2>JavaScript Arrays</h2>

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

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

</body>
</html>
```

Example: array length

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

<h2>JavaScript Arrays</h2>
<p>The length property returns the length of an array.</p>

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

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

Example: void

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


<p>
<a href="javascript:void(0);">
  Useless link
</a>
</p>

<p>
<a href="javascript:void(document.body.style.backgroundColor='red');">
  Click me to change the background color of body to red.
</a>
</p>

</body>
</html>
```

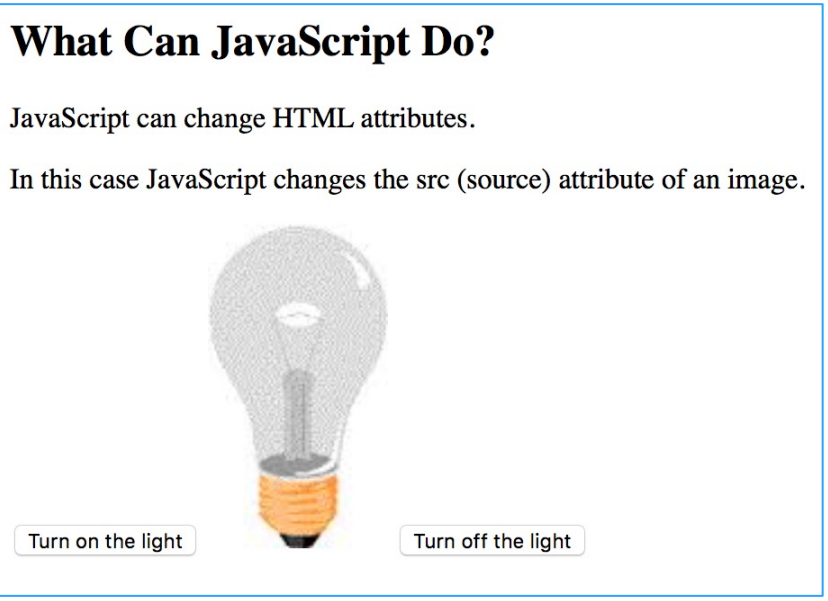
The backgroundColor property sets or returns the background color of an element

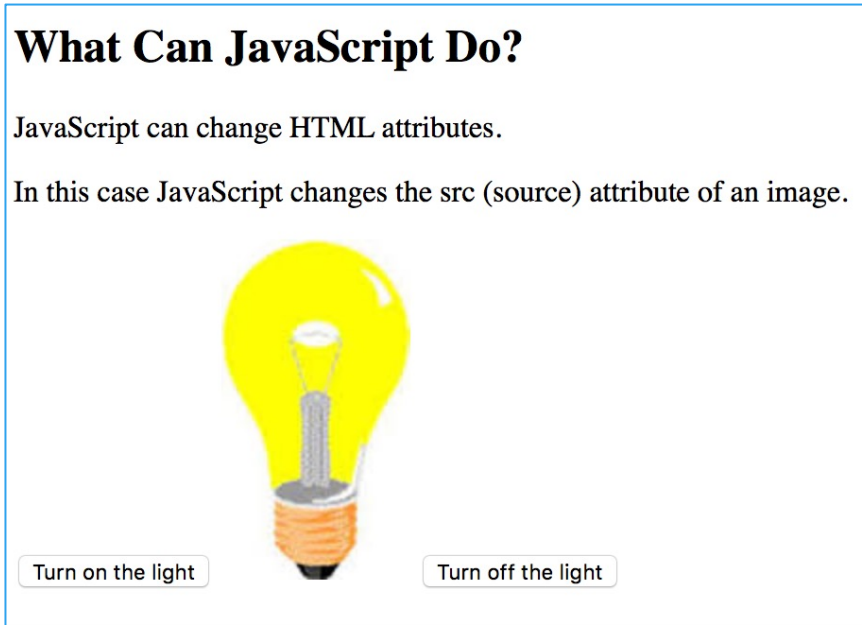
Useless link

Click me to change the background color of body to red.

Example: images (lamp.html)

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

     <h2>What Can JavaScript Do?</h2>
     <p>JavaScript can change HTML attributes.</p>
     <p>In this case JavaScript changes the src (source) attribute of an image.</p>

     <button onclick="document.getElementById('myImage').src='images.jpeg'">Turn on the light</button>

     <img id="myImage" src="images-2.jpeg" style="width:100px">

     <button onclick="document.getElementById('myImage').src='images-2.jpeg'">Turn off the light</button>

    </body>
</html>
```





Oggetti e array

- Gli oggetti in realtà sono **array associativi**
  - strutture composite i cui elementi sono accessibili mediante un indice di tipo stringa (nome) anziché attraverso un indice numerico
- Si può quindi utilizzare anche una sintassi analoga a quella degli array
- Le due sintassi sono del tutto equivalenti e si possono mescolare

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

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

Stringhe

- Non è facile capire esattamente cosa sono le stringhe in JavaScript
- Potremmo dire che mentre in Java sono oggetti che sembrano dati di tipo primitivo in JavaScript sono **dati di tipo primitivo che sembrano oggetti**
- Sono sequenze arbitrarie di caratteri in formato UNICODE a 16 bit e sono immutabili come in Java
- Esiste la possibilità di definire costanti stringa (string literal) delimitate da apici singoli (**'ciao'**) o doppi (**"ciao"**)
- È possibile la concatenazione con l'operatore **+**
- È possibile la comparazione con gli operatori **< > >= <=** e **!=**

Example

- When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2

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

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

<script>
document.getElementById("demo").innerHTML = "2" < "12";
</script>

</body>
</html>
```

false

Stringhe come oggetti?

- Possiamo però invocare metodi su una stringa o accedere ai suoi attributi
- Possiamo infatti scrivere
  - var s = “ciao”;
  - var n = s.length;
  - var t = s.charAt(1);
- Non sono però oggetti e la possibilità di trattarli come tali nasce da due caratteristiche:
  - Esiste un tipo wrapper String che è un oggetto
  - *JavaScript fa il boxing in automatico come C#*
    - quando una variabile di tipo valore necessita essere convertita in tipo riferimento, un oggetto box è allocato per mantenere tale valore

String and number basics

- You can use double or single quotes

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

- Strings have length property

```
"Foobar".length → 6
```

- Numbers can be converted to strings
  - Automatic conversion during concatenations.

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

  - Conversion with fixed precision

```
var n = 123.4567;
var 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

```
var i = parseInt("37 blah");
            //result is 37 - ignores blah

var d = parseFloat("6.02 blah");
            //result is 6.02 - ignores blah
```

Core string methods

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

```
"Hello".charAt(1); → "e"
"Hello".indexOf("o"); → 4  //returns -1 if no match
"hello".substring(1,3); → "el"
"Hello".toUpperCase(); → "HELLO"
```

- Methods that use regular expressions
  - match, replace, search, split https://www.w3schools.com/js/js_string_methods.asp
- HTML methods
  - anchor, big, bold, fixed, fontcolor, fontsize, italics, link, small, strike, sub, sup

```
"test".bold().italics().fontcolor("red")
→ '<font color="red"><i><b>test</b></i></font>'
```

  - These are technically nonstandard methods, but supported in all major browsers

JavaScript data types

```
var length = 16;                                    // Number
var lastName = "Johnson";                           // String
var cars = ["Saab", "Volvo", "BMW"];                // Array
var x = {firstName:"John", lastName:"Doe"};         // Object
```

```
var x;                  // Now x is undefined
var x = 5;              // Now x is a Number
var x = "John";         // Now x is a String
```

```
var carName = "Volvo XC60";   // Using double quotes
var carName = 'Volvo XC60';   // Using single quotes
```

```
var answer = "It's alright";              // Single quote inside double quotes
var answer = "He is called 'Johnny'";     // Single quotes inside double quotes
var answer = 'He is called "Johnny"';     // Double quotes inside single quotes
```

The typeof operator

- You can use the JavaScript typeof operator to find the type of a JavaScript variable:

```
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 (if myCar is not declared)
typeof null                   // Returns object
```

Tipi valore e tipi riferimento

- Si può distinguere fra **tipi valore** e **tipi riferimento**
  - Numeri e booleani sono tipi valore
  - Array e Oggetti sono tipi riferimento

- Per le stringhe abbiamo una situazione incerta
- Pur essendo un tipo primitivo si comportano come un tipo riferimento

Funzioni

- Una funzione è un frammento di codice JavaScript che viene definito una volta e usato in più punti
  - Ammette parametri che sono privi di tipo
  - Restituisce 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**
- Una funzione può essere assegnata ad una variabile

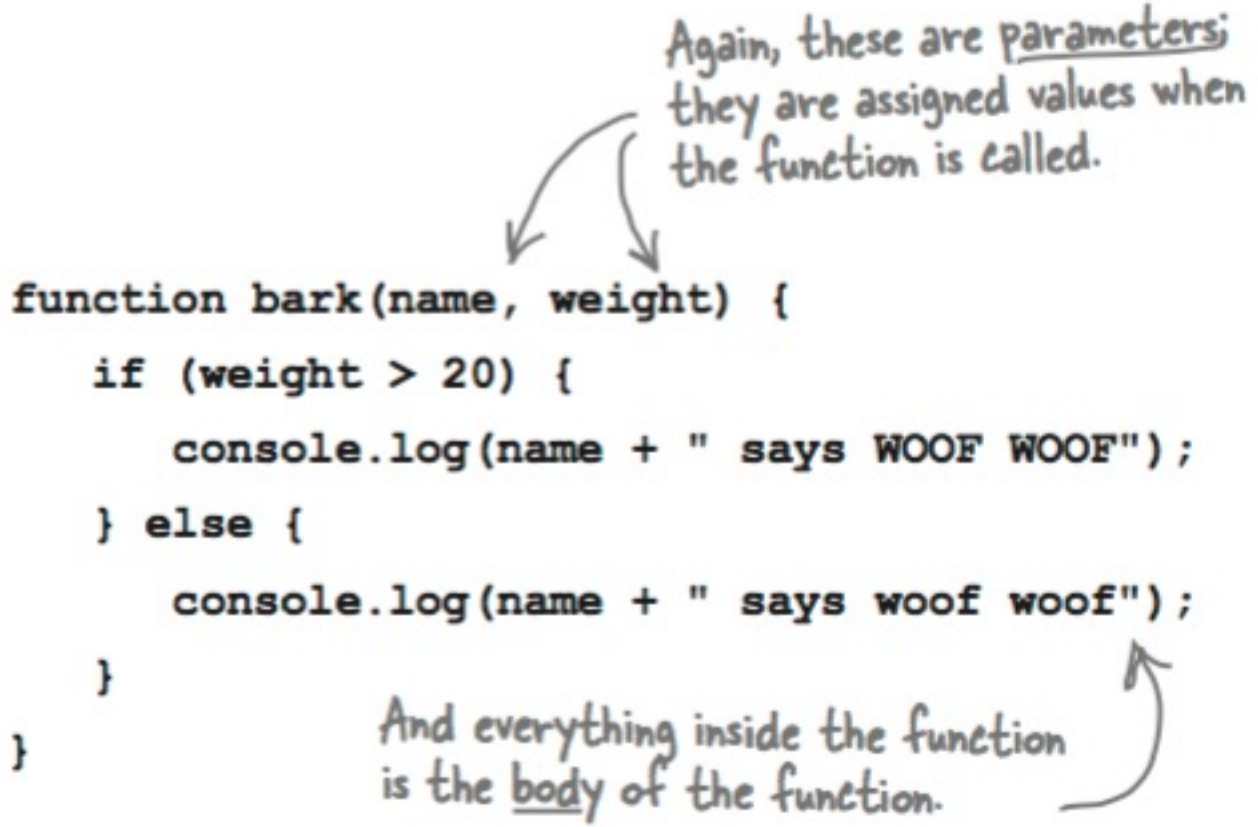
```
function sum(x,y)
{
  return x+y;
}
```

```
var s = sum(2,4);
```

Arrow functions allow us to write shorter function syntax
https://www.w3schools.com/js/js_arrow_function.asp

```
sum = (x, y) => x + y;
```

Example: function definition



Example: call a function

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Bark</title>
  </head>
  <body>
  </body>
  <script>
function bark(name, weight) {
   if (weight > 20) {
      console.log(name + " says WOOF WOOF");
   } else {
      console.log(name + " says woof woof");
   }
}
bark("rover", 23);
bark("spot", 13);
bark("spike", 53);
bark("lady", 17);
  </script>
</body>
</html>
```

Test the bark function

We just did this...

...so do this next.

```
}
}
bark("rover", 23);
bark("spot", 13);
bark("spike", 53);
bark("lady", 17);
```

```
rover says WOOF WOOF
spot says woof woof
spike says WOOF WOOF
lady says woof woof
>
```

Costanti funzione e costruttore Function

- Esistono **costanti funzione** (function literal) che permettono di definire una funzione e poi di assegnarla ad una variabile con una sintassi decisamente inusuale:

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

- Una funzione può essere anche creata usando un costruttore denominato **Function** (le funzioni sono quindi equivalenti in qualche modo agli oggetti)

```
var sum =
  new Function("x","y","return x+y;");
```

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

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;
}
```

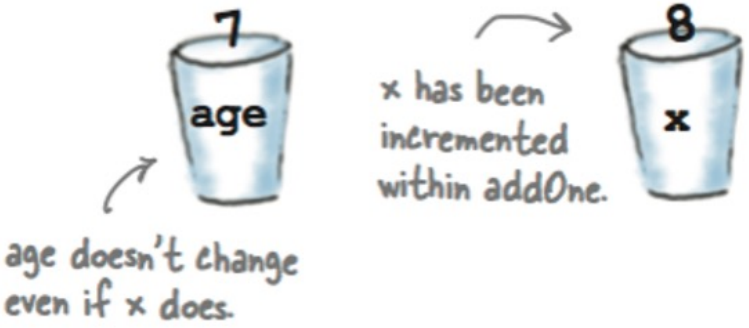
3 Now let's call the function addOne, pass it the variable age as the argument. The value in age is copied into the parameter x.

```
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

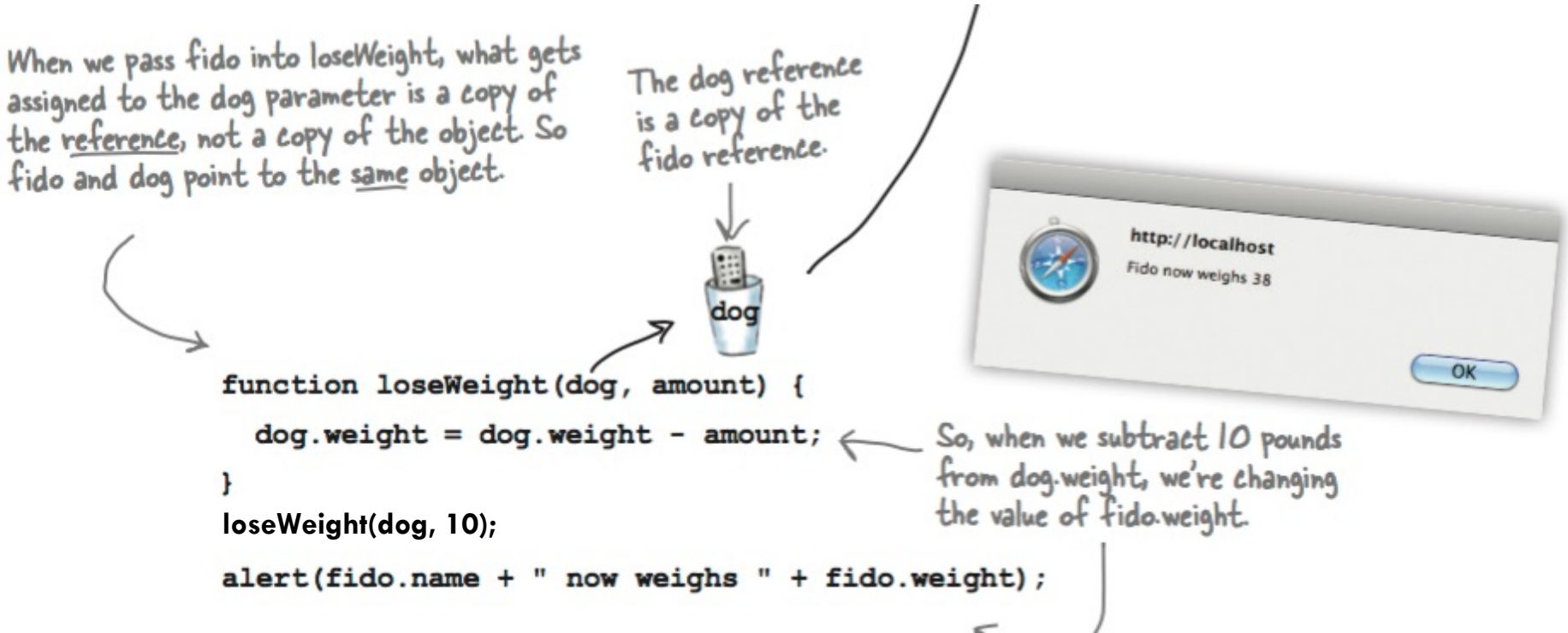
- When an object is assigned to a variable, that variable holds a ***reference*** to the object, not the object itself
- When you call a function and pass it an object, you're passing **the *object reference,*** not the object itself

- 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:

var dog = {name: "Fido", weight: 48, ...}



Variable scope

- **Global variables live as long as the page**
  - With the **var** clause or without (with an assignment)

- **Local variables typically disappear when your function ends**
  - Local variables are created (with the **var** clause) when your function is first called and live until the function returns (with a value or not)
  - Without the **var** clause (with an assignment) the variable is defined globally

- **Block variables disappear when the block ends**
  - With the **var** clause it lives until the end of the block
  - Without the **var** clause (with an assignment) the variable is defined globally

Metodi

- Quando una funzione viene assegnata ad una proprietà di un oggetto viene chiamata metodo dell'oggetto
- 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
- Costruiamo un oggetto con 2 attributi e un metodo:

```
var o = new Object();
o.x = 15;
o.y = 7;
o.tot = function() { return this.x + this.y; }
alert(o.tot());
```

Costruttori

- Un costruttore è una funzione che ha come scopo quello di costruire un oggetto
- Se viene invocato con **new** riceve l'oggetto appena creato e può aggiungere proprietà e metodi
- L'oggetto da costruire è accessibile con la parola chiave **this**
- In qualche modo definisce il tipo di un oggetto

```
function Rectangle(w, h)
{
  this.w = w;
  this.h = h;
  this.area = function()
    { return this.w * this.h; }
  this.perimeter = function()
    { return 2*(this.w + this.h);
}
 }
```

```
var r = new Rectangle(5,4);
alert(r.area());
```

Proprietà e metodi statici

- JavaScript ammette l'esistenza di proprietà e metodi statici con lo stesso significato di Java
- Sono associati al costruttore
- Per esempio, se abbiamo definito il costruttore Circle() che serve per creare oggetti di tipo cerchio, possiamo aggiungere l'attributo PI in questo modo:

```
function Circle(r)
{
  this.r = r;
}
Circle.PI = 3.14159;
```

- In JavaScript esiste l'oggetto **Math** che definisce solo metodi statici corrispondenti alle varie funzioni matematiche (es. **Math.PI**)

Function example: AreaAndDistance

- **You can put your functions anywhere in your JavaScript file**
- JavaScript doesn't care if your functions are declared before or after you use them

```
function setup(width, height) {
    centerX = width/2;
    centerY = height/2;
}
function computeDistance(x1, y1, x2, y2) {
    var dx = x1 - x2;
    var dy = y1 - y2;
    var d2 = (dx * dx) + (dy * dy);
    var d = Math.sqrt(d2);
    return d;
}
function circleArea(r) {
    var area = Math.PI * r * r;
    return area;
}
var width = 5;
var height = 4;
setup(width, height);
var area = circleArea(2);
var distance = computeDistance(0, 0, centerX, centerY);
alert("Area: " + area);
alert("Distance: " + distance);
```

global variables

local variables

global variables

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, function
  - Istruzioni speciali: break, continue, return

For statement

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

for (var i = 0; i < scores.length; i++) {

    var output = "Bubble solution #" + i +
                 " score: " + scores[i];

    console.log(output);

}
```

All we've done is update where we increment the loop variable with the post-increment operator.

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***) ***statement***

```
var x;
var mycars = new Array();
mycars[0] = "Panda";
mycars[1] = "Uno";
mycars[2] = "Punto";
mycars[3] = "Clio";
for (x in mycars)
{
  document.write(mycars[x]+"<br />");
}
```

Example: for

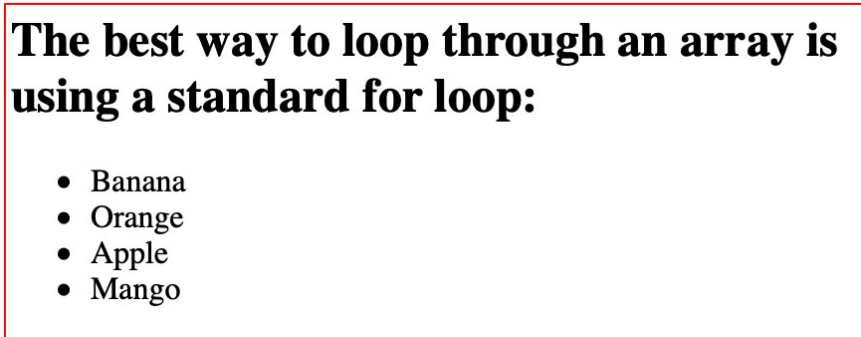
```
<!DOCTYPE html>
<html>
<body>
<p>The best way to loop through an array is using a standard for loop:</p>

<p id="demo"></p>
<script>
        var fruits, text, fLen, i;

        fruits = ["Banana", "Orange", "Apple", "Mango"];
        fLen = fruits.length;
        text = "<ul>";
        for (i = 0; i < fLen; i++) {
                text += "<li>" + fruits[i] + "</li>";
        }
        text += "</ul>";
        document.getElementById("demo").innerHTML = text;
</script>
</body>
```

The best way to loop through an array is using a standard for loop:

- Banana
- Orange
- Apple
- Mango

Example: for-in

```
<!DOCTYPE html>
<html>
<body>
<p>The best way to loop through an array is using a standard for loop:</p>

<p id="demo"></p>
<script>
        var fruits, text, fLen, i;

        fruits = ["Banana", "Orange", "Apple", "Mango"];
        fLen = fruits.length;
        text = "<ul>";
        for (x in fruits) {
                text += "<li>" + fruits[x] + "</li>";
        }
        text += "</ul>";
        document.getElementById("demo").innerHTML = text;
</script>
</body>
```



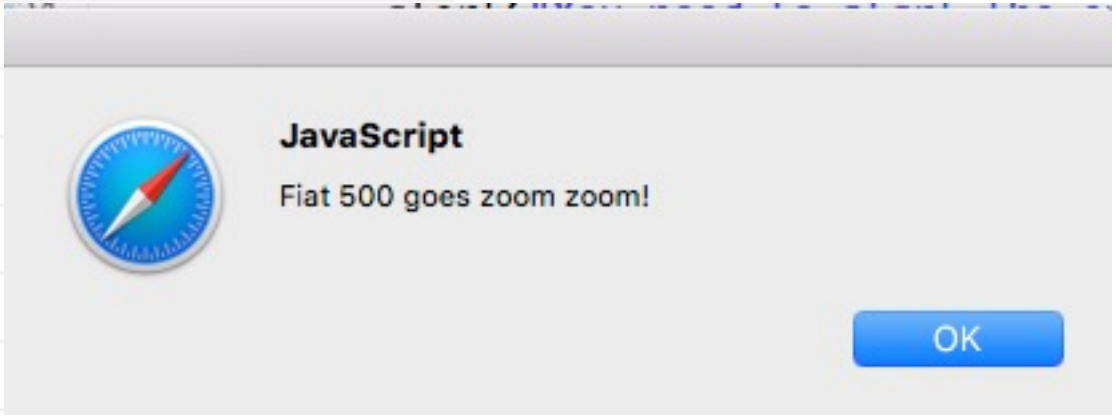
Esempio: for-in on an object (carProps.html)

```
<script>

var fiat = {
    make: "Fiat",
    model: "500",
    year: 1957,
    color: "Medium Blue",
    passengers: 2,
    convertible: false,
    mileage: 88000,
    started: false,

    start: function() {
        this.started = true;
    },

    stop: function() {
        this.started = false;
    },
```

Esempio: for-in on an object (2)

```
    drive: function() {
        if (this.started) {
            alert(this.make + " " +
                    this.model + " goes zoom zoom!");
        } else {
            alert("You need to start the engine first.");
        }
    }
};

fiat.start();
fiat.drive();

for (prop in fiat) {
    console.log(prop + ": " + fiat[prop]);
}

</script>
```

```
make: Fiat
model: 500
year: 1957
color: Medium Blue
passengers: 2
convertible: false
mileage: 88000
started: true
start: function () {
    this.started = true;
}
stop: function () {
    this.started = false;
}
drive: function () {
    if (this.started) {
        alert(this.make + " " +
            this.model + " goes zoom zoom!");
    } else {
        alert("You need to start the engine first.");
    }
}
```



L'oggetto globale e funzioni predefinite

- In JavaScript esiste un oggetto **globale implicito**
- **Tutte le variabili e le funzioni definite in una pagina appartengono all'oggetto globale**
- Possono essere utilizzate senza indicare questo oggetto
- Questo oggetto espone anche 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 [,radix])** converte la stringa str in un intero (in base radix – opzionale)
  - **parseFloat(str)** converte la stringa str in un numero

Array operations

- In JavaScript, arrays can have methods
  - Not functions to which you *pass* arrays, but methods *of* arrays

```
var nums = [1,2,3];
nums.reverse(); → [3,2,1]

[1,2,3].reverse(); → [3,2,1]
```

- Most important methods:
  - push, pop, concat
  - sort
  - forEach
  - map
  - filter
  - reduce

Push, pop, concat

- Push

```
var nums = [1,2,3];
nums.push(4);
nums; → [1,2,3,4]
```

- Pop

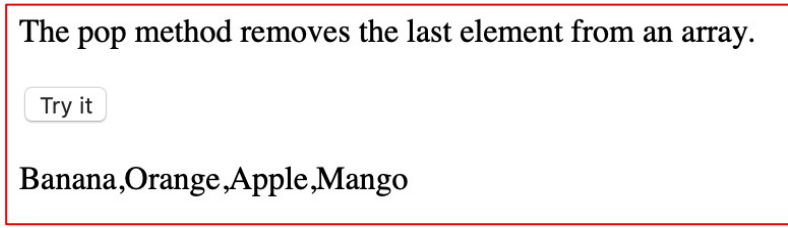
```
var val = nums.pop();
val;  → 4
nums;  → [1,2,3]
```

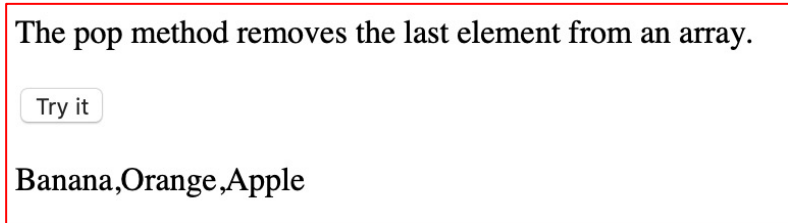
- Concat

```
nums;  → [1,2,3]
var nums2 = nums.concat([4,5,6]);
nums2;  → [1,2,3,4,5,6]
```

Example: pop

```
<!DOCTYPE html>
<html>
<body>
        <p>The pop method removes the last element from an array.</p>
        <button onclick="myFunction()">Try it</button>
        <p id="demo"></p>
<script>
        var fruits = ["Banana", "Orange", "Apple", "Mango"];
        document.getElementById("demo").innerHTML = fruits;
        function myFunction() {
                fruits.pop();
                document.getElementById("demo").innerHTML = fruits;
        }
</script>

</body>
</html>
```





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]
```

Foreach

- Calls function on each element of array. Cannot break "loop" partway through
  - Lacks option to run in parallel that Java has
- Examples

```
[1,2,3].forEach(function(n) { alert(n); });
```

  - Pops up alert box in page 3 times showing each number

```
[1,2,3].forEach(alert);
```

  - Same as above
  - Summing an array (but reduce can also be used)

```
var nums = [1,2,3];
var sum = 0;
nums.forEach(function(n) { sum += n; });
sum; → 6
```

Map

- Calls function on each element, then accumulates result array of each of the outputs. Returns new array; does not modify original array
  - Like the Java map method, but not as powerful since the JavaScript version does not support lazy evaluation or parallel operations
- Example

```
function square(n) { return(n * n); }
[1,2,3].map(square); → [1,4,9]
```

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
  - Like the Java filter method, but not as powerful since the JavaScript version does not support lazy evaluation or parallel operations
- Example

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

Reduce

- Takes function and starter value. Each time, passes accumulated result and next array element through function, until a single value is left
  - Like the Java reduce method, but not as powerful since the JavaScript version does not support lazy evaluation or parallel operations
- Examples

```
function add(n1,n2) { return(n1 + n2); }
function multiply(n1,n2) { return(n1 * n2); }
function bigger(n1,n2) { return(n1 > n2 ? n1 : n2); }

var nums = [1,2,3,4];
var sum = nums.reduce(add, 0);    //10
var product = nums.reduce(multiply, 1);  //24
var max = nums.reduce(bigger, -Number.MAX_VALUE); //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] //makes copy of array
    ```
- Reverse
  - Reverses array (returns it, but also changes original)

    ```
    [1,2,3].reverse(); → [3,2,1]
    ```
- Indexof
  - Finds index of matching element

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

Exercize: What does it do?

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

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

<script>
    var t = [10,10,7,7,5,5,5,2,-1,-1,-2];
    function whatDoesItDo(arr) {
      return arr.sort().filter(function(elem, pos) {
                    return arr.indexOf(elem) == pos;
                  });
    };

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


</body>
</html>
```

-1,-2,10,2,5,7

Splice

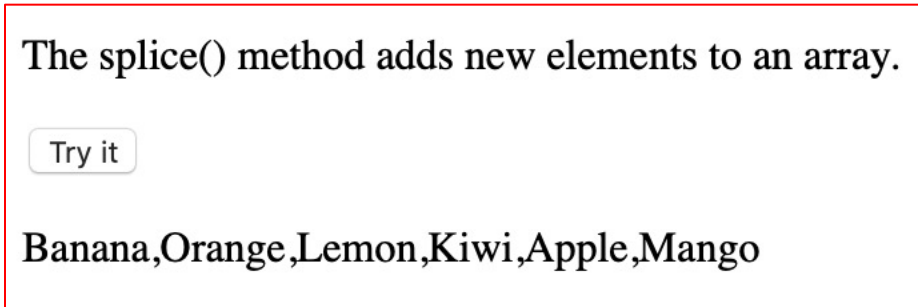
- The **splice** method can be used to add new items to an array:

```
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>
        <p>The splice() method adds new elements to an array.</p>
        <button onclick="myFunction()">Try it</button>

        <p id="demo"></p>
<script>
        var fruits = ["Banana", "Orange", "Apple", "Mango"];
        document.getElementById("demo").innerHTML = fruits;
        function myFunction() {
                fruits.splice(2, 0, "Lemon", "Kiwi");
                document.getElementById("demo").innerHTML = fruits;
        }
</script>
</body>
</html>
```



JavaScript classes (ECMAScript 6)

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  static hello() {
    return "Hello!!";
  }
  present(x) {
    return x + ", I have a " + this._carname;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show(x) {
    return this.present(x) + ', it is a ' + this.model;
  }
}

var mycar = new Car("Ford");
document.getElementById("demo").innerHTML += mycar.present("Hello") + "<br>";
document.getElementById("demo").innerHTML += Car.hello() + "<br>";

mycar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML += mycar.show("Hello") + "<br>";
document.getElementById("demo").innerHTML += mycar.carname + "<br>";
mycar.carname = "Volvo";
document.getElementById("demo").innerHTML += mycar.carname + "<br>";
```

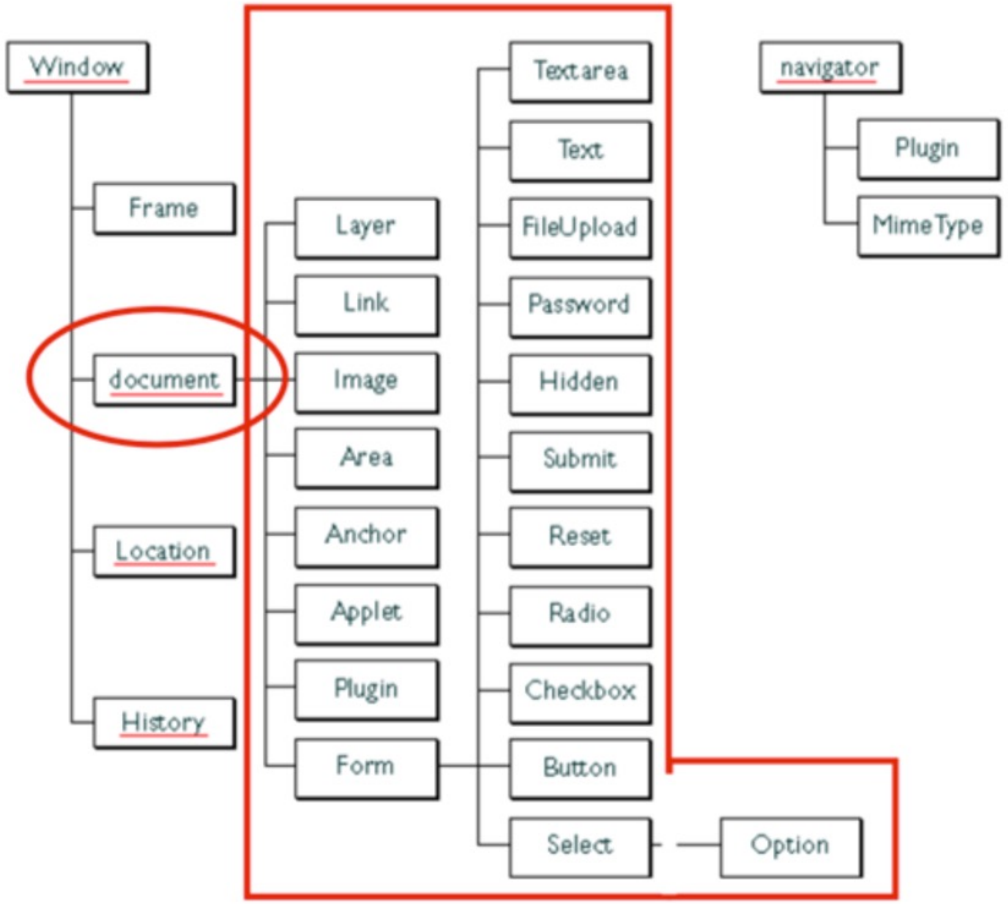
- A class is a type of function, but we use the keyword **class**, and the properties are assigned inside a **constructor()** method

Hello, I have a Ford
Hello!!
Hello, I have a Ford, it is a Mustang
Ford
Volvo

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



Costruzione dinamica della pagina

- La più semplice modalità di utilizzo di JavaScript consiste nell'inserire nel corpo della pagina script che generano dinamicamente parti della pagina
- Bisogna tener presente che questi script vengono eseguiti solo una volta durante il caricamento della pagina e quindi **non si ha interattività con l'utente**
- La **pagina corrente** è rappresentata dall'oggetto **document**
  - Per scrivere nella pagina si utilizzano i metodi **document.write()** e **document.writeln()**

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):

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

<script>
var w = window.innerWidth
|| document.documentElement.clientWidth
|| document.body.clientWidth;

var h = window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;

var x = document.getElementById("demo");
x.innerHTML = "Browser inner window width: " + w + ", height: " + h + ".";
</script>
```

Browser inner window width: 692, height: 782.

Browser inner window width: 497, height: 506.

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")'/>

- *È possibile inserire più istruzioni in sequenza, ma è meglio definire delle funzioni*
- *Attenzione. È necessario alternare doppi apici e apice singolo*

```
<input type="button" value="Apriti sesamo!"
        onClick="window.open('myDoc.html','newWin')">
<input type="button" value="Apriti Unisa!"
        onClick="window.open('http://www.unisa.it','newWin')">
```

Events-1

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

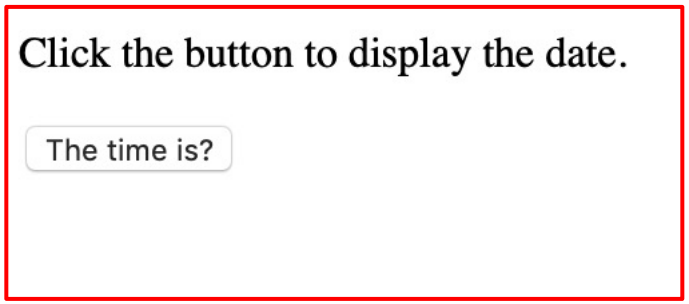
Evento	Applicabilità	Occorrenza	Event handler
Abort	Immagini	L'utente blocca il caricamento di un'immagine	onAbort
Blur	Finestre e tutti gli elementi dei form	L'utente toglie il focus a un elemento di un form o a una finestra	onBlur
Change	Campi di immissione di testo o liste di selezione	L'utente cambia il contenuto di un elemento	onChange
Click	Tutti i tipi di bottoni e i link	L'utente 'clicca' su un bottone o un link	onClick onDblClick
DragDrop	Finestre	L'utente fa il drop di un oggetto in una finestra	onDragDrop onDrag, onDrop
Error	Immagini, finestre	Errore durante il caricamento	onError
Focus	Finestre e tutti gli elementi dei form	L'utente dà il focus a un elemento di un form o a una finestra	onFocus
KeyDown	Documenti, immagini, link, campi di immissione di testo	L'utente preme un tasto	onKeyDown
KeyPress	Documenti, immagini, link, campi di immissione di testo	L'utente digita un tasto (pressione + rilascio)	onKeyPress
KeyUp	Documenti, immagini, link, campi di immissione di testo	L'utente rilascia un tasto	onKeyUp

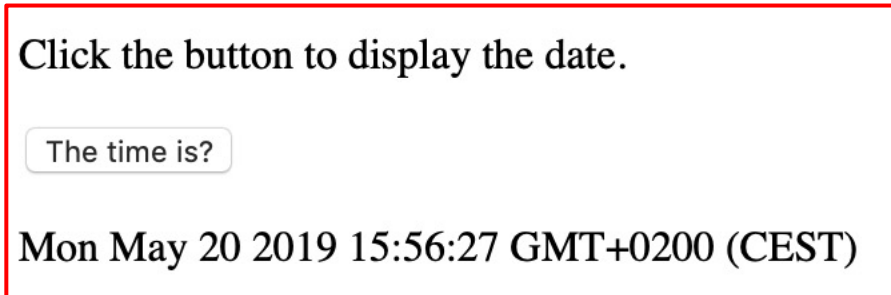
Events-2

Evento	Applicabilità	Occorrenza	Event handler
Load	Corpo del documento	L'utente carica una pagina nel browser	onLoad
MouseDown	Documenti, bottoni, link	L'utente preme il bottone del mouse	onMouseDown
MouseMove	Di default nessun elemento	L'utente muove il cursore del mouse	onMouseMove
MouseOut	Mappe, link	Il cursore del mouse esce fuori da un link o da una mappa	onMouseOut
MouseOver	Link	Il cursore passa su un link	onMouseOver
MouseUp	Documenti, bottoni, link	L'utente rilascia il bottone del mouse	onMouseUp
Move	Windows	La finestra viene spostata	onMove
Reset	Form	L'utente resetta un form	onReset
Resize	Finestre	La finestra viene ridimensionata	onResize
Select	Campi di immissione di testo (input e textarea)	L'utente seleziona il campo	onSelect
Submit	Form	L'utente sottomette il form	onSubmit
Unload	Corpo del documento	L'utente esce dalla pagina	onUnload

Example: onClick event

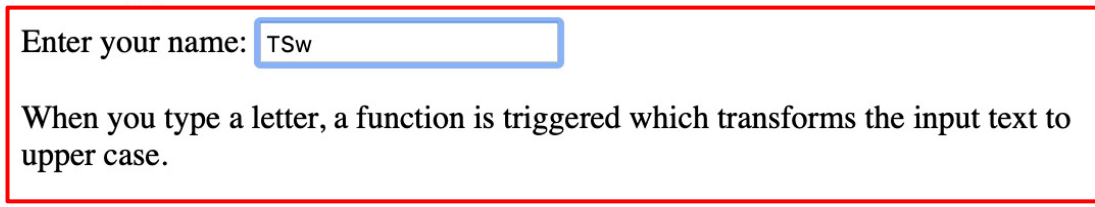
```
<!DOCTYPE html>
<html>
<body>
        <p>Click the button to display the date.</p>
        <button onclick="displayDate()">The time is?</button>

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

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

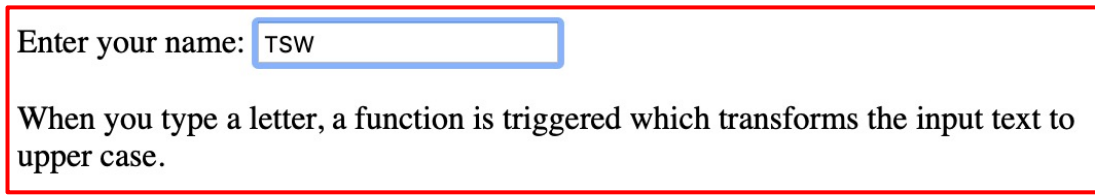




Example: onKeyUp event

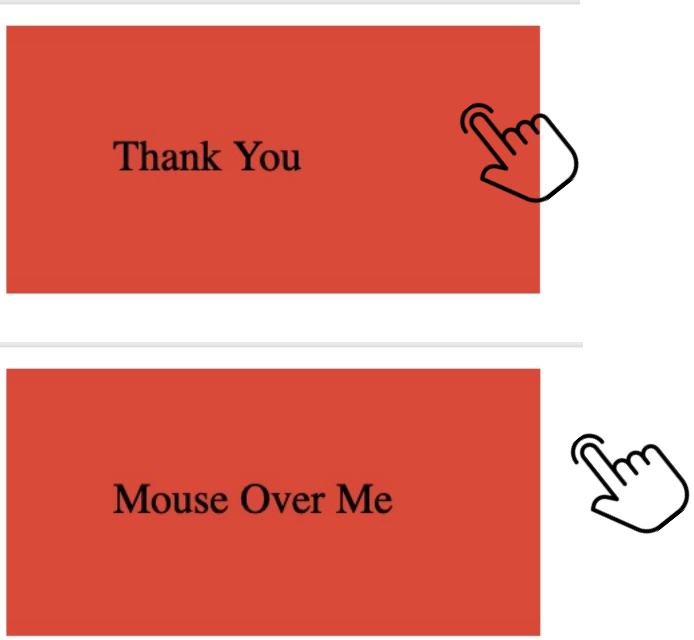
```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
    var x = document.getElementById("fname");
      x.value = x.value.toUpperCase();
}
</script>
</head>
<body>

Enter your name: <input type="text" id="fname"
onkeyup="myFunction()">
<p>When you type a letter, a function is triggered which
transforms the input text to upper case.</p>

</body>
</html>
```





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>
```



Example: setInterval and clearInterval

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

<p>A script on this page starts this clock:</p>
<p id="demo" onclick="stopTimer()"></p>

<script>
var myVar = setInterval(function(){ myTimer() }, 1000);

function myTimer() {
    var d = new Date();
    var t = d.toLocaleTimeString();
    document.getElementById("demo").innerHTML = t;
}

function stopTimer() {
   clearInterval(myVar);
}
</script>

</body>
</html>
```

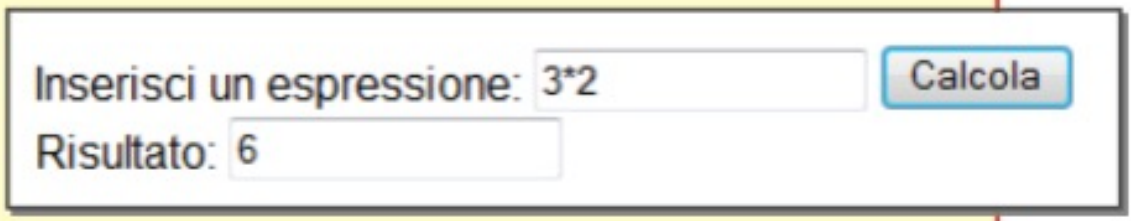
A script on this page starts this clock:

5:40:47 PM

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

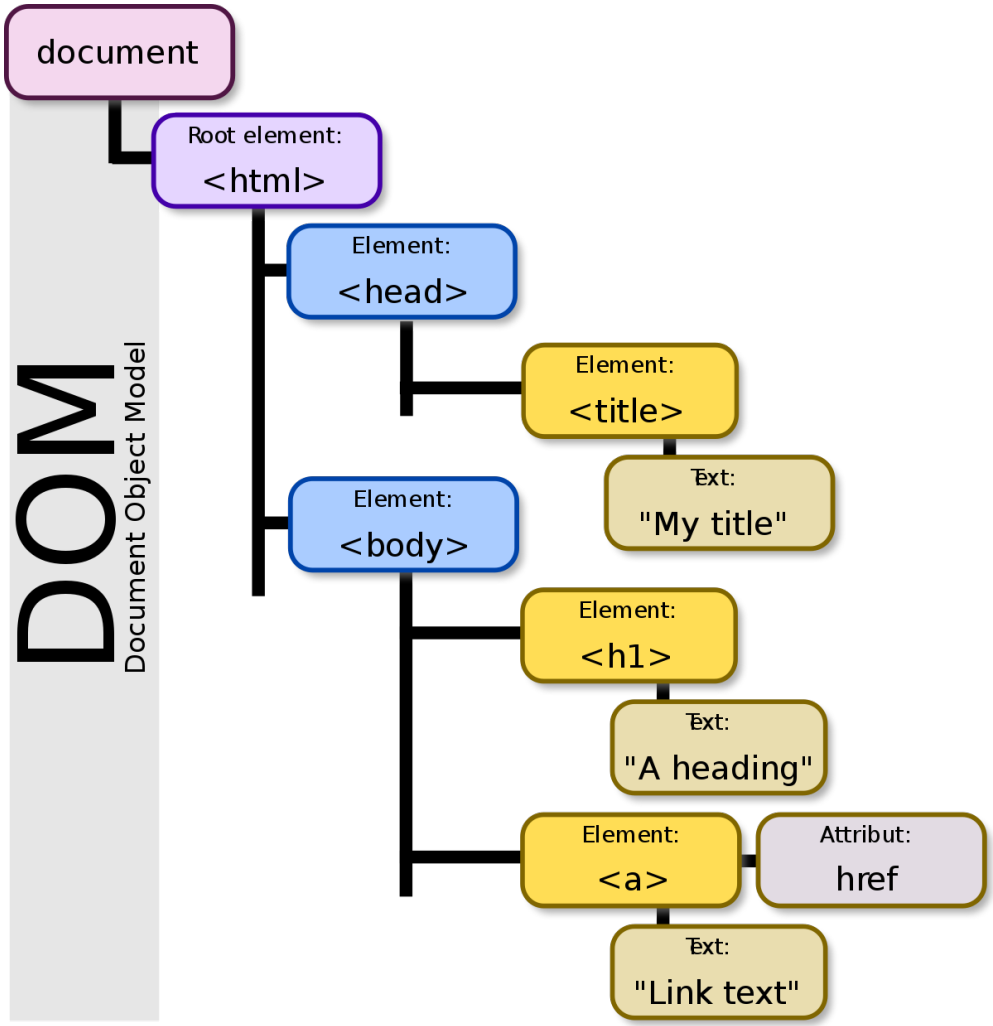


riferimento al form che contiene il controllo

Inserire nel campo espressione: **window.location.href = 'https:\\www.unisa.it';**

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>
```



document is always at the top. document is a special part of the tree that you can use in JavaScript to get access to the entire DOM.

document

document is also like the root of an upside down tree.

html

These are like the branches of the tree.

head

body

meta

title

script

h1

div id="entry1"

My blog

My blog

h2

p

p

Great day bird watching

Today I saw three...

I took a couple of photos...

These are like the leaves of the tree (because there are no elements inside them, just text).

The DOM includes the content of the page as well as the elements. (We don't always show all the text content when we draw the DOM, but it's there).

Esplorare il DOM: document

- Il punto di partenza per accedere al Documento Object Model (DOM) della pagina è l'oggetto **document**
- **document** espone 4 collezioni di oggetti che rappresentano gli elementi di primo livello:
  - **anchors[]** **(es., <a name="ancora">…)**
  - **forms[]**
  - **images[]**
  - **links[]** **(es., …)**
- L'accesso agli elementi delle collezioni può avvenire per indice (ordine di definizione nella pagina) o per nome (attributo name dell'elemento):

  document.links[0]

  document.links["*nomelink*"]
- Può essere scritta anche come **document.*nomelink***

  document.links[0].href → **http://www.unisa.it**

  document.links["lnk"].href → **http://www.unisa.it**

Document

- Metodi:
  - **getElementById()**: restituisce un riferimento al primo oggetto della pagina avente l'id specificato come argomento
  - **write()**: scrive un pezzo di testo nel documento
  - **writeln()**: come write() ma aggiunge un a capo
  - …
- Proprietà:
  - **bgColor**: colore di sfondo
  - **fgColor**: colore di primo piano
  - **lastModified**: data e ora di ultima modifica
  - **cookie**: tutti i cookies associati al document
    - rappresentati da una stringa di coppie: nome-valore
  - **title**: titolo del documento
  - **URL**: url del documento

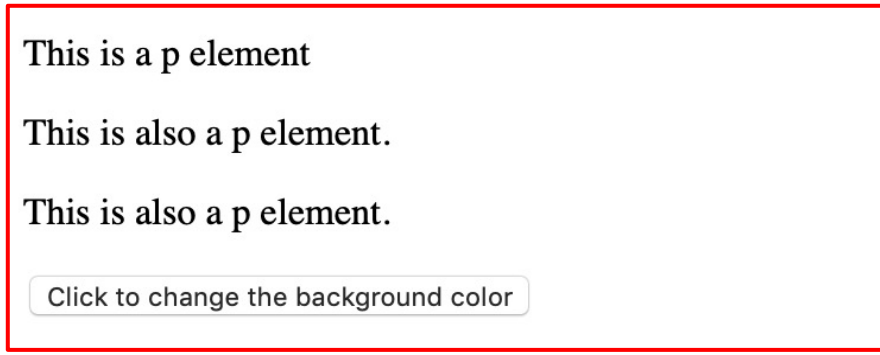
Document (2)

- **getElementById()** returns the element that has the ID attribute with the specified value
- **getElementsByClassName()** returns a NodeList containing all elements with the specified class name
- **getElementsByName()** returns a NodeList containing all elements with a specified name
- **getElementsByTagName()** returns a NodeList containing all elements with the specified tag name

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

Example: getElementsByTagName method

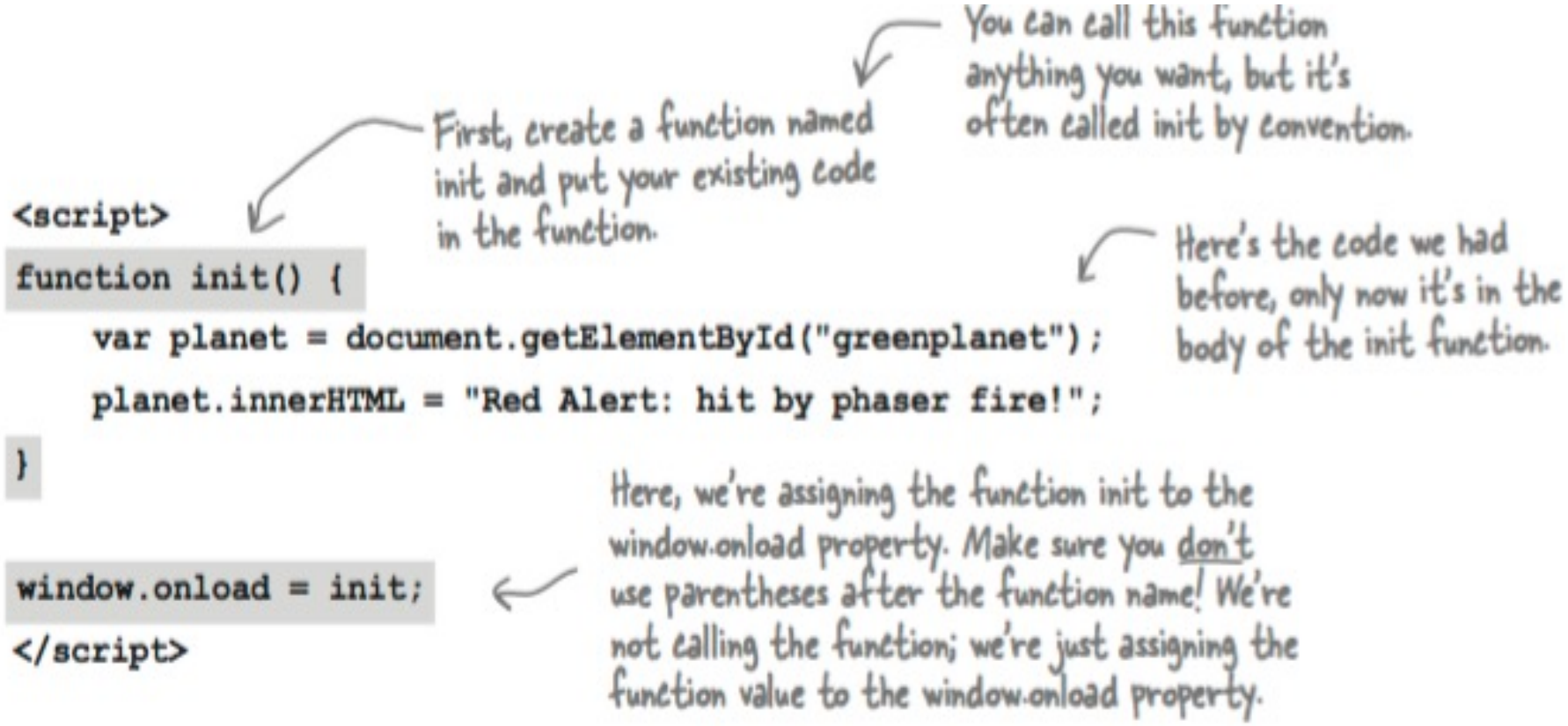
```
<!DOCTYPE html>
<html>
<body>

<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're dealing with the DOM it's important to execute your code only *after* the page is *fully loaded*.** If you don't, there's a good chance the DOM won't be created by the time your code executes
- First **create a function** that has the code you'd like executed *once the page is fully loaded*
- Next, you take the window object, and assign the function to its onload property

onLoad (the "page is loaded" event)



Example: onLoad event

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

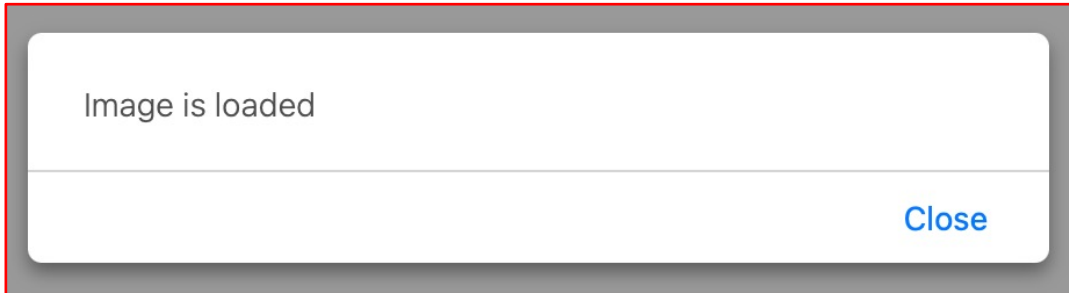
```
<!DOCTYPE html>
<html>
<body onload="myFunction()">

<h1>Hello World!</h1>

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

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

Example: onload on a tag

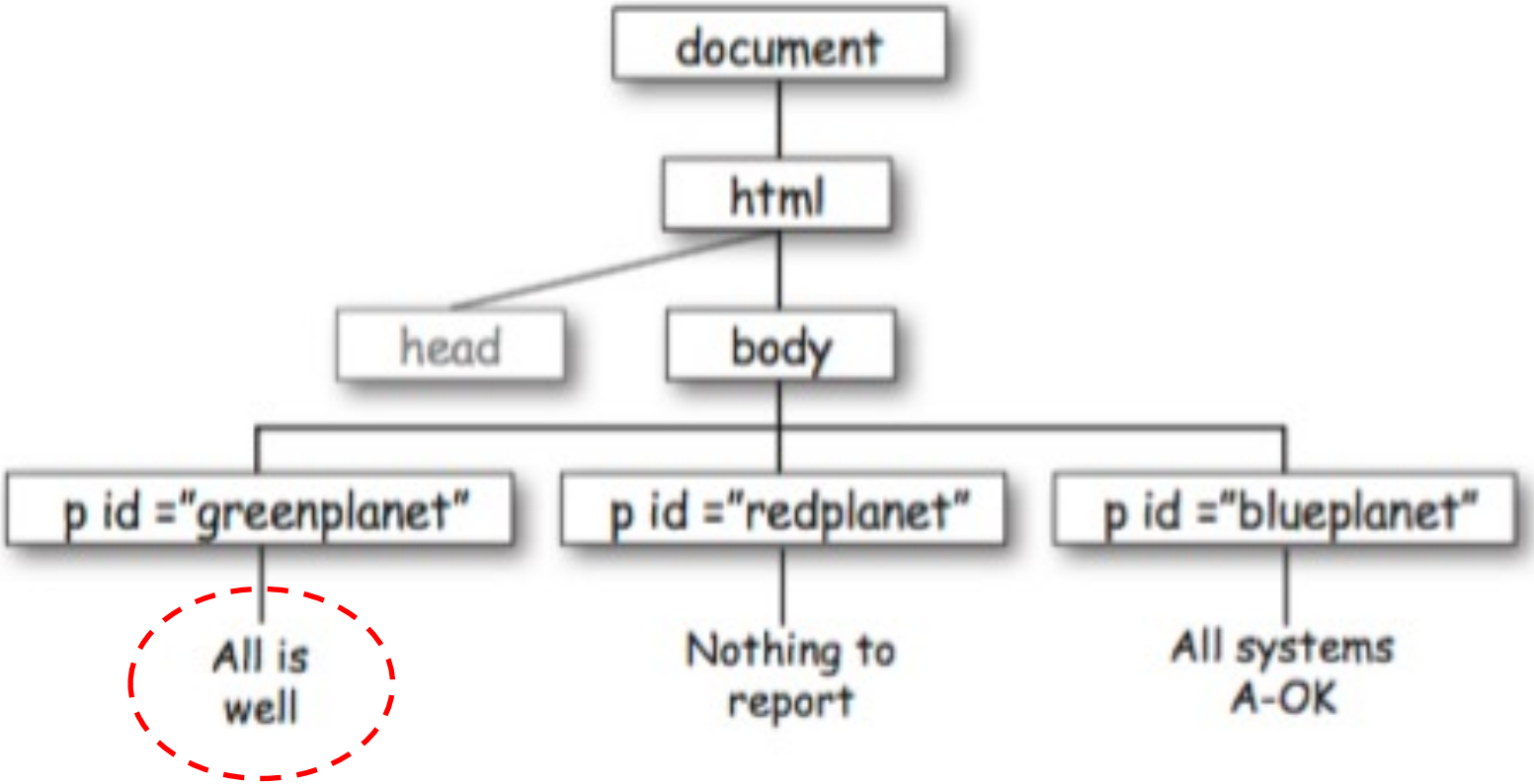
- Using onload on an <img> element
- Alert "Image is loaded" immediately after an image has been loaded

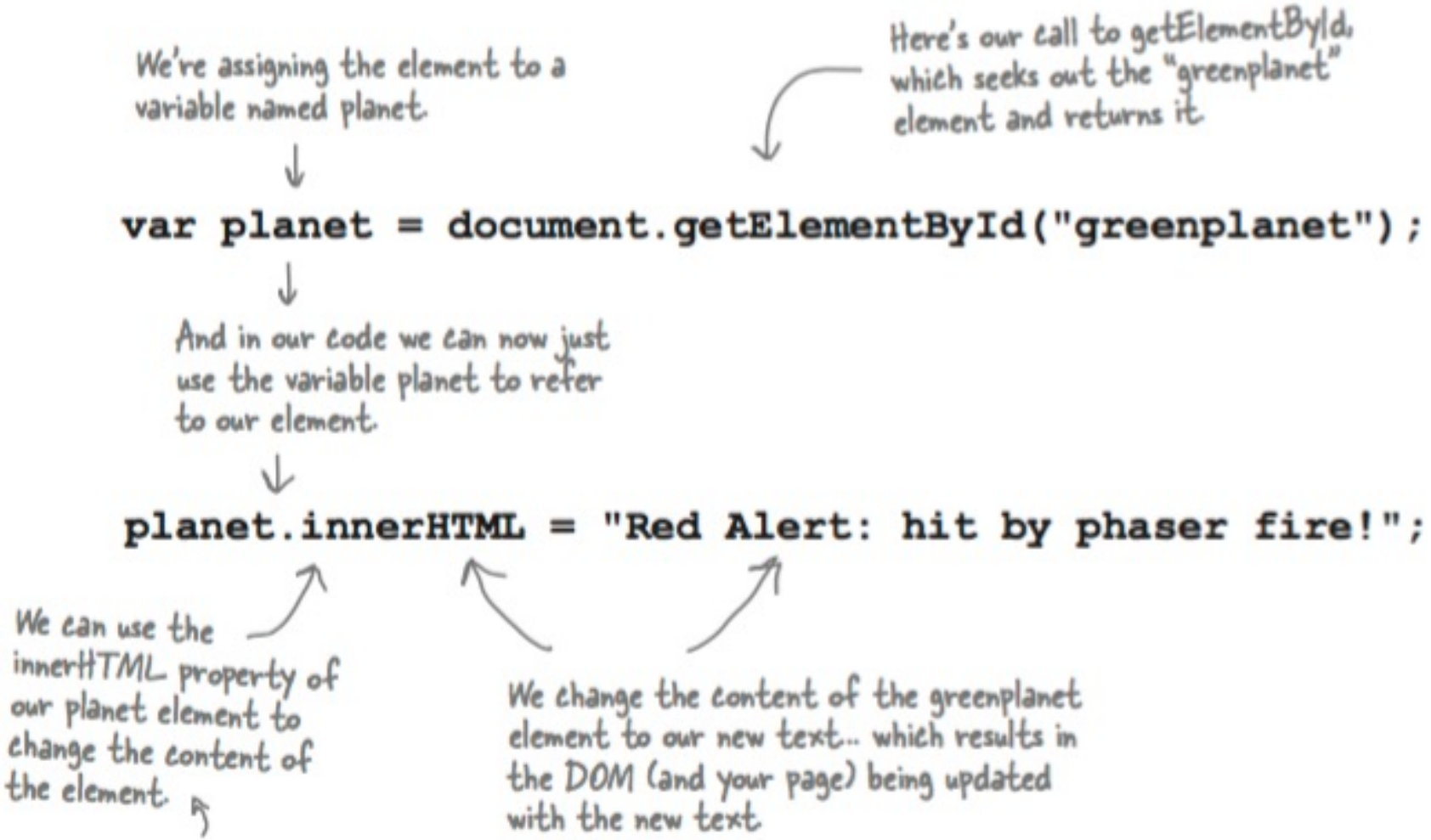
```
<!DOCTYPE html>
<html>
<body>

<img src="w3javascript.gif" onload="loadImage()" width="100" height="132">

<script>
function loadImage() {
  alert("Image is loaded");
}
</script>

</body>
</html>
```

soon

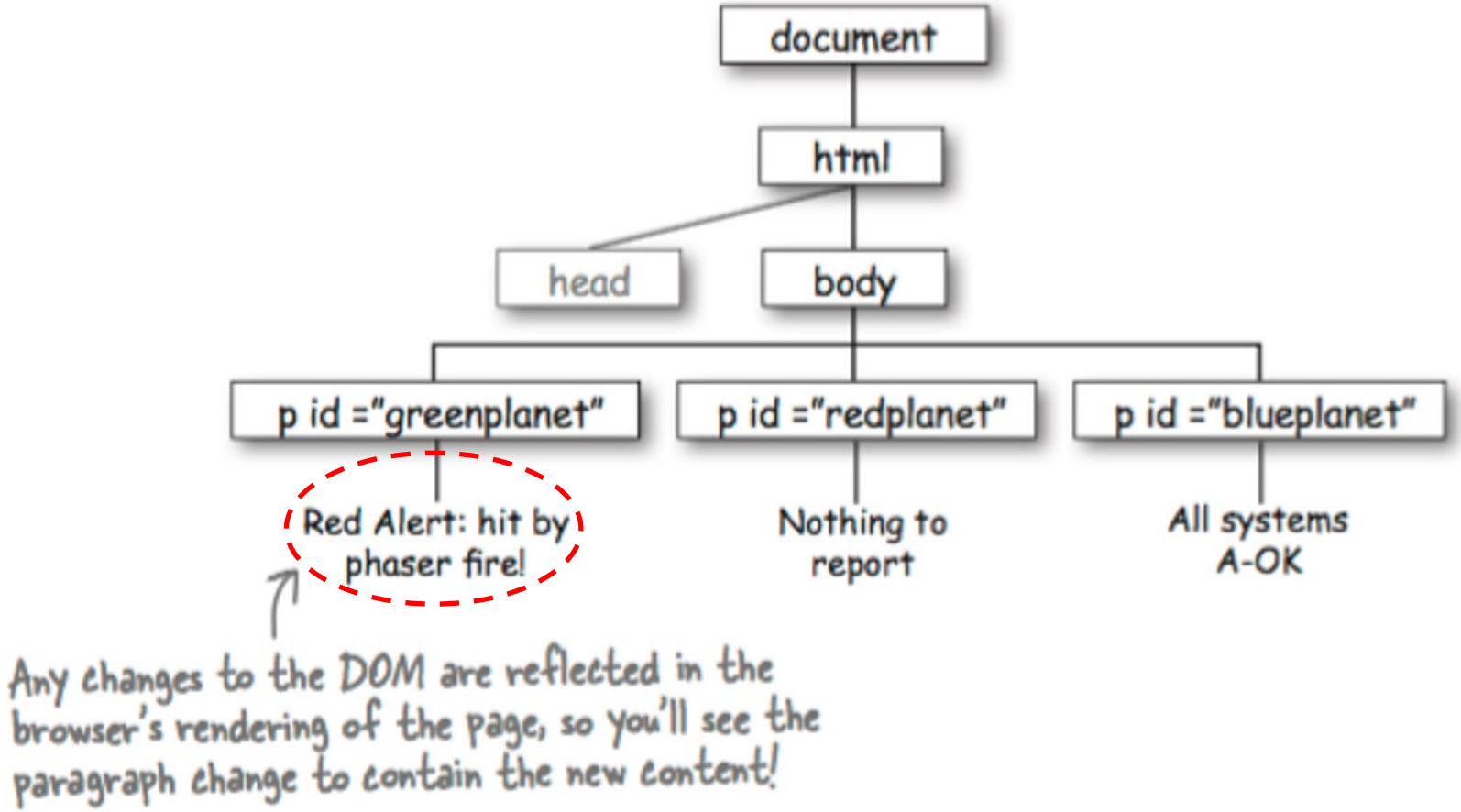


Example: change the DOM




We're assigning the element to a variable named planet.
Here's our call to getElementById, which seeks out the "greenplanet" element and returns it.
var planet = document.getElementById("greenplanet");
And in our code we can now just use the variable planet to refer to our element.
planet.innerHTML = "Red Alert: hit by phaser fire!";
We can use the innerHTML property of our planet element to change the content of the element.
We change the content of the greenplanet element to our new text... which results in the DOM (and your page) being updated with the new text.

Example: DOM has been changed

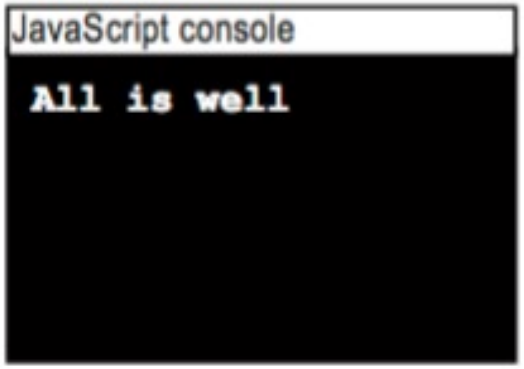


Finding the inner HTML

- 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!"

JavaScript console

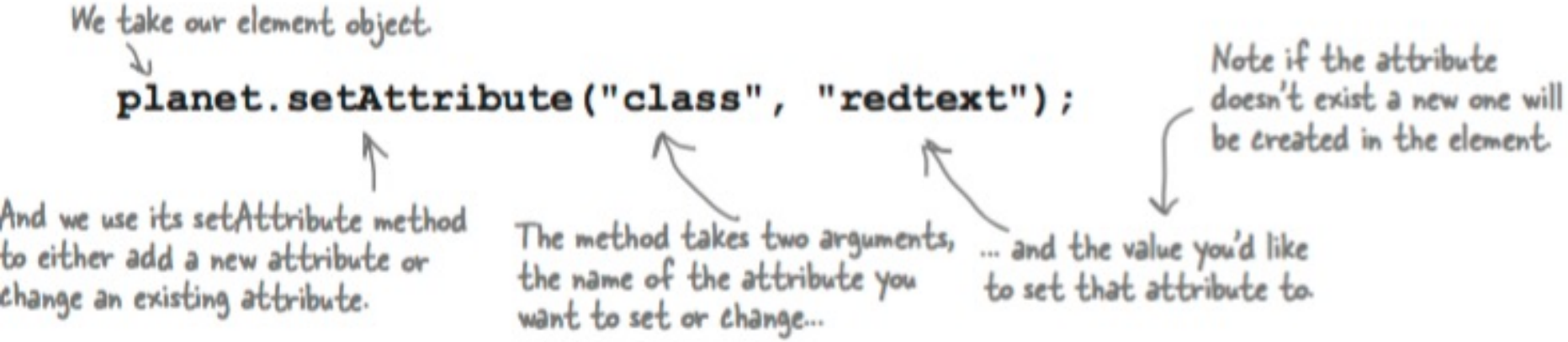
```
Red Alert: hit by
phaser fire!
```

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

And the web page changes too!

How to set an attribute with setAttribute

- Element objects have a method named **setAttribute** that you can call to set the value of an HTML element's attribute




```
element.setAttribute("class", "redtext");
```
Bad performance!!!

```
element.className = "redtext";
```

```
element.classList.add("redtext");
```
Multiple classes

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

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

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>
```

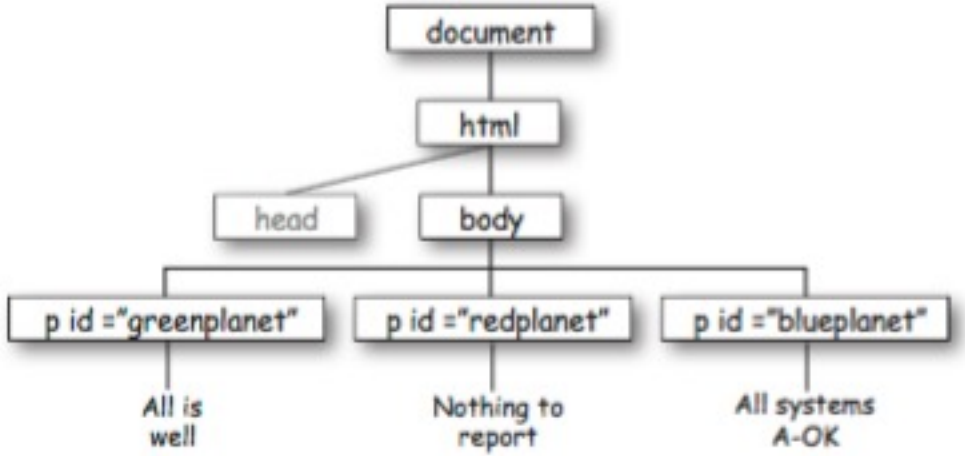
We've got the redtext class included here so when we add "redtext" as the value for the class attribute in our code, it turns the text red.

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!

Before...

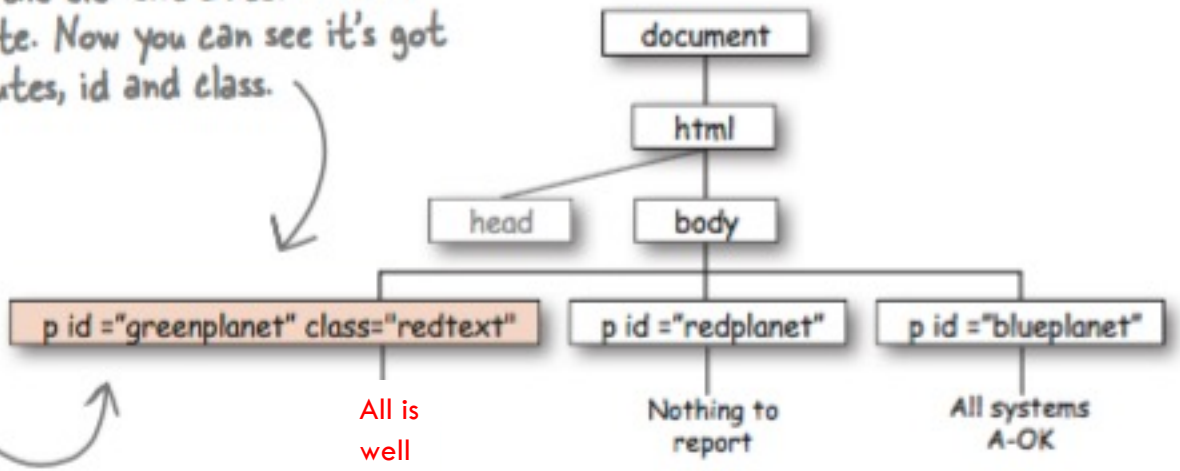
Here's the element before we call the setAttribute method on it. Notice this element already has one attribute, id.



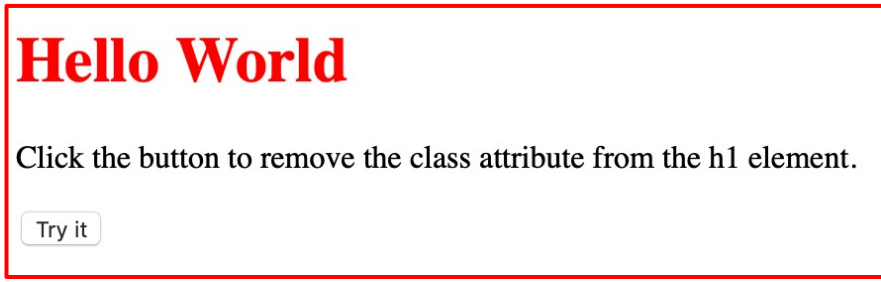
And After

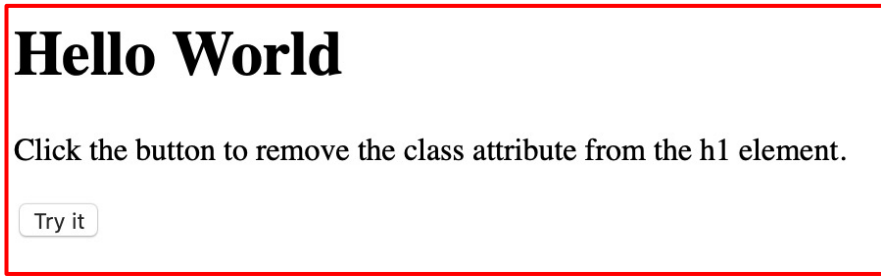
And here's the element after we call setAttribute. Now you can see it's got two attributes, id and class.

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

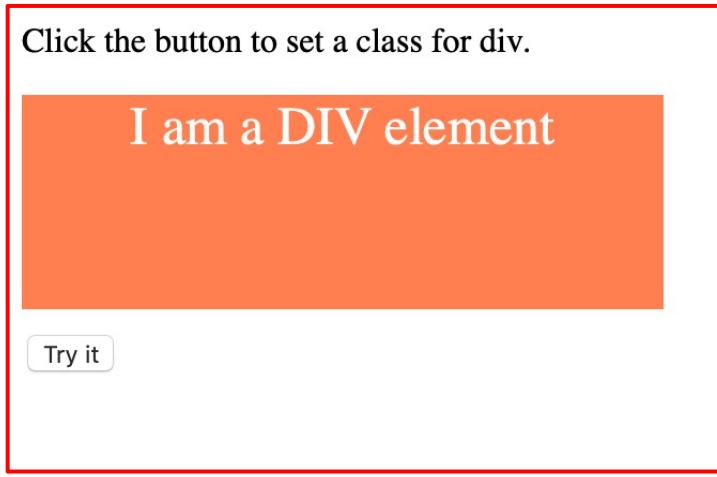





```
<!DOCTYPE html>
<html>
<head>
<style>
.democlass { color: red; }
</style>
</head>
<body>

<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 CSS class

```
<!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>
```

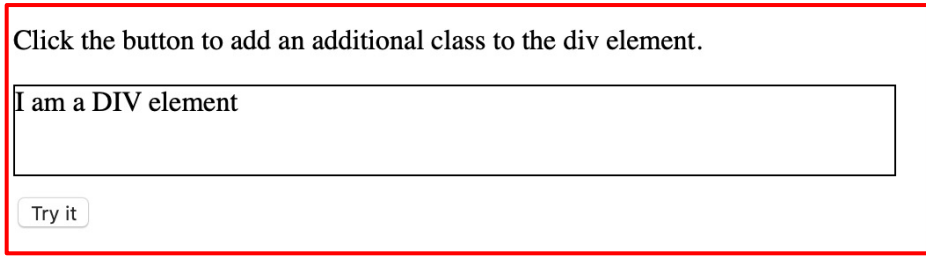


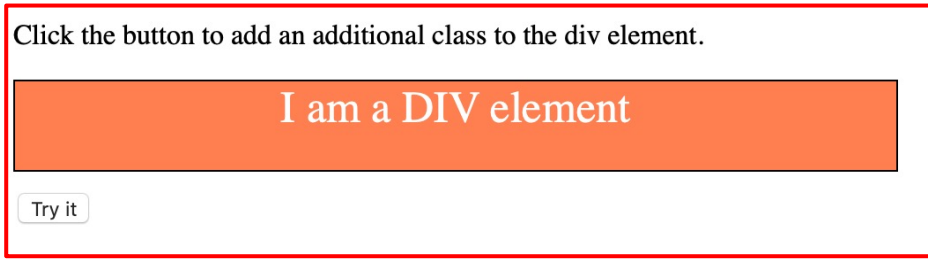
JavaScript can change HTML styles (CSS)

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

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<script>
function myFunction() {
    var x = document.getElementById("demo");
    if(x) {
        x.style.fontSize = "20px";
        x.style. color = "red";
    }
}
</script>

</body>
</html>
```

JavaScript can add CSS 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>
```





DOM is good for…

Get elements from the DOM

- Use document.getElementById…
- Use tag names, class names and attributes 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

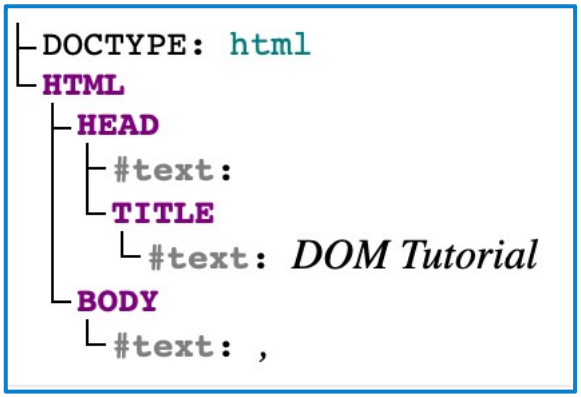
- You can also remove elements from the DOM by taking a parent element and removing any of its children
- You'll see the element removed in your browser window as soon as it is deleted from the DOM

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** (or **value**)

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>
```

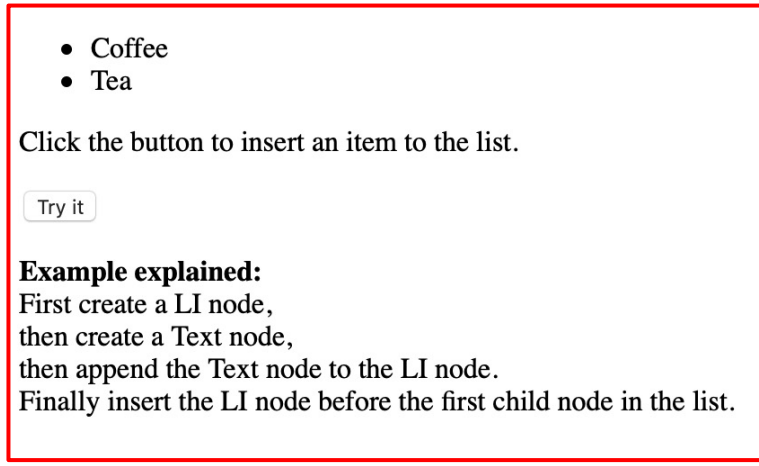
```
▼ <html>
  ▼ <head>
      <title id="myTitle">Old title: DOM Tutorial New title: DOM Hacking</title>
    </head>
```

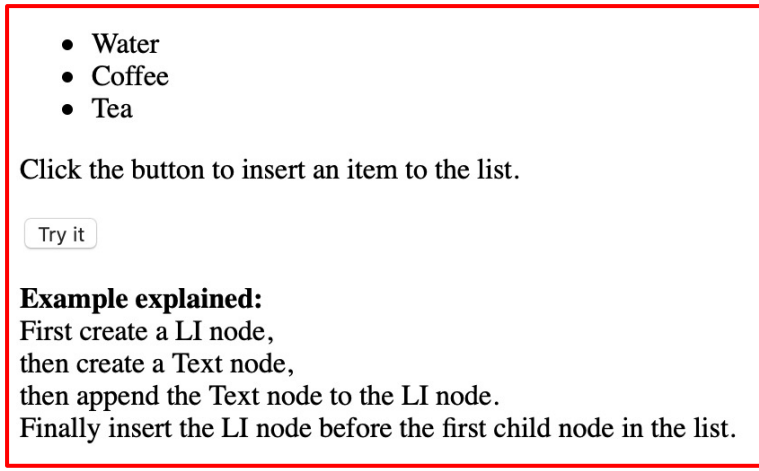
Example: creating new HTML elements

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

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para,child);
</script>

</body>
</html>
```

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>
```



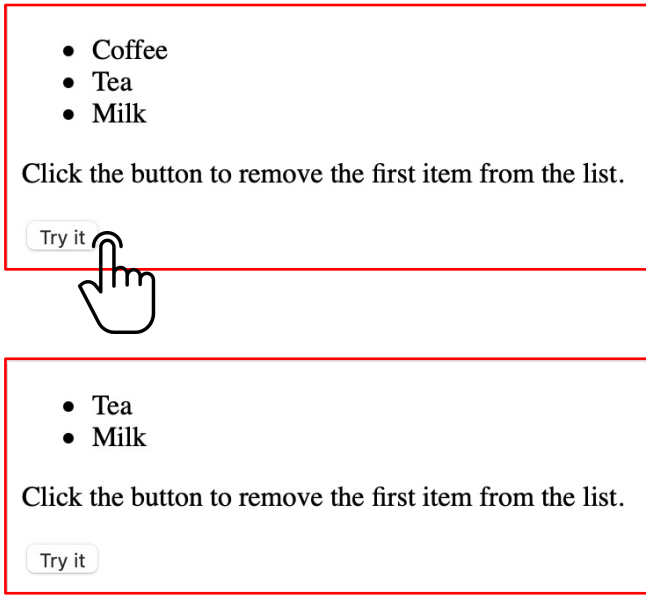


Example: removing existing HTML elements

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

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>

</body>
</html>
```

Example: removing existing HTML elements (2)

```
<!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>
```

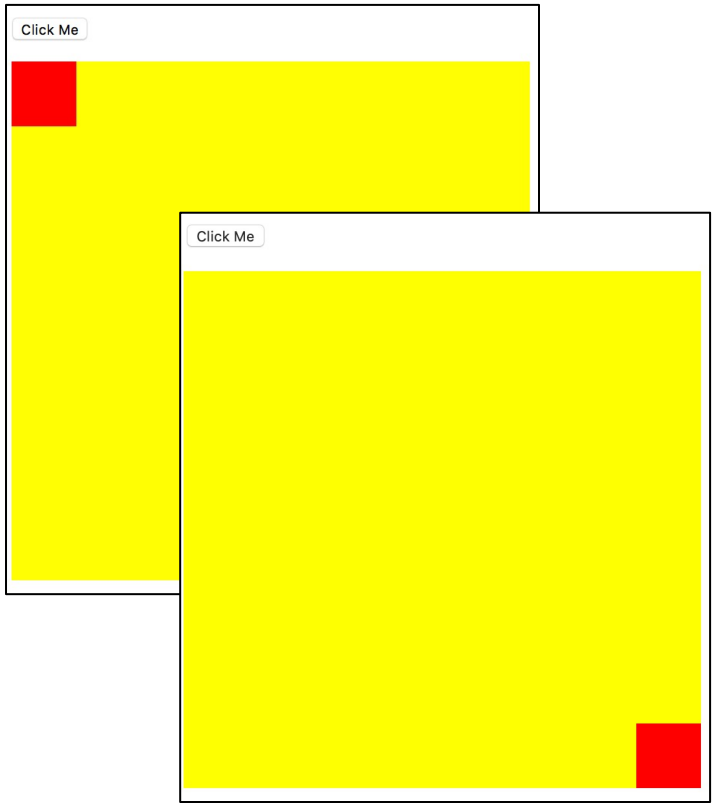


Example: change HTML attributes

```
<!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>
```

JavaScript animation

```
<!DOCTYPE html>
<html>
<style>
#container {
  width: 400px; height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px; height: 50px;
  position: absolute;
  background-color: red;
}
</style>
<body>

<p><button onclick="myMove()">Click Me</button></p>

<div id ="container">
<div id ="animate"></div>
</div>

<script>
function myMove() {
  var elem = document.getElementById("animate");
  var pos = 0;
  var id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + 'px';
      elem.style.left = pos + 'px';
    }
  }
}
</script>
</body>
</html>
```

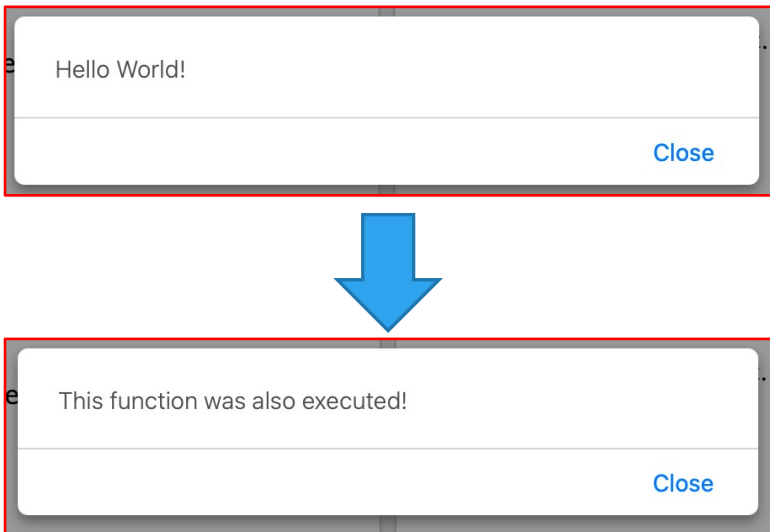


Example: addEventListener method

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

<p>This example uses the addEventListener() method to attach a click event to a button.</p>

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

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

<script>
document.getElementById("myBtn").addEventListener("click", displayDate);

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

</body>
</html>
```

Example: addEventListener method (2)

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

<p>This example uses the addEventListener() method to add two click events to
the same button.</p>

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

<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);

1 function myFunction() {
    alert ("Hello World!");
}

2 function someOtherFunction() {
    alert ("This function was also executed!");
}
</script>

</body>
</html>
```



Form

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

  document.*nomeForm*

  document.forms[n]

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

  document.*nomeForm.nomeElemento*

  document.forms[n].elements[m]

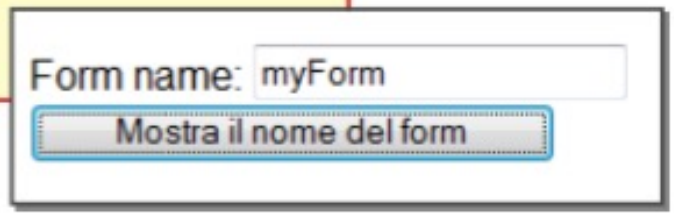
  document.forms["*nomeForm*"].elements[*"nomeElem"*]
- Ogni elemento ha una proprietà **form** che permette di accedere al form che lo contiene (vedi esempio “calcolatrice” precedente **this.form** del campo input)

Form (2)

- Per ogni elemento del form esistono proprietà corrispondenti ai vari attributi:

  id, name, value, type, className...

```
<form name="myForm">
  Form name:
  <input type="text" name="text1" value="test">
  <br/>
  <input name="button1" type="button"
    value="Mostra il nome del form"
    onclick="document.myForm.text1.value=
                document.myForm.name">
</form>
```



In alternativa potevamo scrivere:

```
onclick="this.form.text1.value=
                this.form.name">
```

Form (3)

- Proprietà:
  - **action**: riflette l'attributo action
  - **elements**: vettore contenente gli elementi della form
  - **length**: numero di elementi nella form
  - **method**: riflette l'attributo method
  - **name**: nome del form
  - **target**: riflette l'attributo target (dove mostrare la risposta dopo il submit)
- Metodi:
  - **reset()**: resetta il form
  - **submit()**: esegue il submit
- Eventi:
  - **onReset**: quando il form viene resettato
  - **onSubmit**: 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”>**

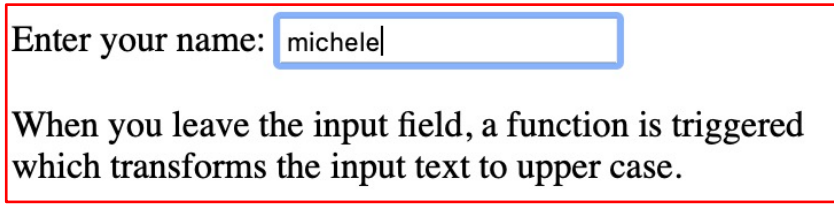
```
var checkbox = document.createElement('input');
checkbox.type = "checkbox";
checkbox.name = "name";
checkbox.value = "value";
checkbox.id = "id";
//form container (reference to a form node)
container.appendChild(checkbox);

var label = document.createElement('label')
label.htmlFor = "id";
label.appendChild(
      document.createTextNode('text….'));
container.appendChild(label);
```

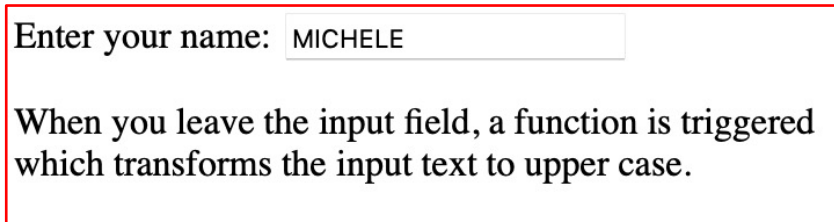
Elementi comuni ai vari controlli

- Proprietà:
  - **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:
  - **onBlur** quando il controllo perde il focus
  - **onFocus** quando il controllo prende il focus
  - **onClick** quando l'utente clicca sul controllo

Example: onBlur event

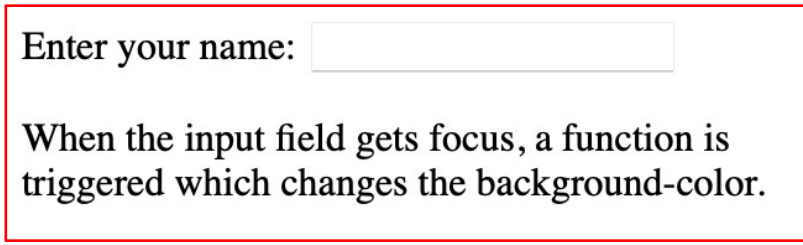
```
<!DOCTYPE html>
<html>
    <body>

      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>
```

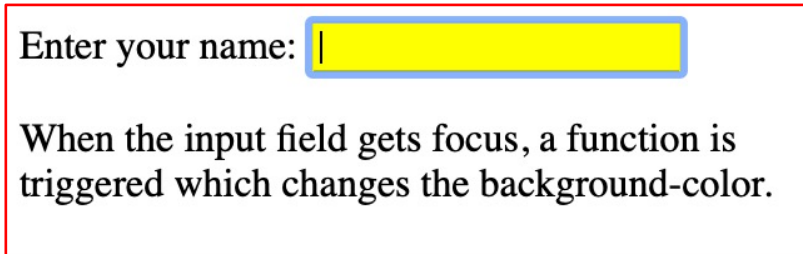




Example: onFocus event

```
<!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>
```





Oggetti text e password

- Proprietà (get/set):
  - **defaultValue** valore di default
  - **disabled** disabilitazione / abilitazione del campo
  - **maxLength** numero massimo di caratteri
  - **readOnly** sola lettura / lettura e scrittura
  - **size** dimensione del controllo
- Metodi:
  - **select()** seleziona una parte di testo

Oggetti checkbox e radio

- Proprietà (get/set):
  - **checked**: dice se il box e spuntato
  - **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
  - Semplifica le applicazioni server side *(non fidarsi troppo)*
  - 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 - 1

```
<head>
  <script type="text/javascript">
    function qty_check(item, min, max)
    {
      returnVal = false;
      if (parseInt(item.value) < min) ||
         (parseInt(item.value) > max)
        alert(item.name+"deve essere fra "+min+" e "+max);
      else returnVal = true;
      return returnVal;
    }
    function validateAndSubmit(theForm)
    {
      if (qty_check(theform.quantità,0,999))
      { alert("Ordine accettato"); return true; }
      else
      {  alert("Ordine rifiutato"); return false; }
    }
  </script>
</head>
```

Esempio di validazione - 2

```
<body>
  <form name="widget_order"
   action="lwapp.html" method="post">
    Quantità da ordinare
    <input type="text" name="quantità"
       onchange="qty_check(this,0,999)">
    <br/>
    <input type="submit" value="Trasmetti l'ordine"
      onclick="validateAndSubmit(this.form)">
  </form>
</body>
```

```
<form name="widget_order"
 action="lwapp.html" method="post"
  onSubmit="return qty_check(this.quantità, 0, 999)">
  ...
  <input type="submit" />
  ...
</form>
```

Example: Validation (checkValidity method)

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

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

    </body>
</html>
```

checkValidity: returns true if an input element contains valid data

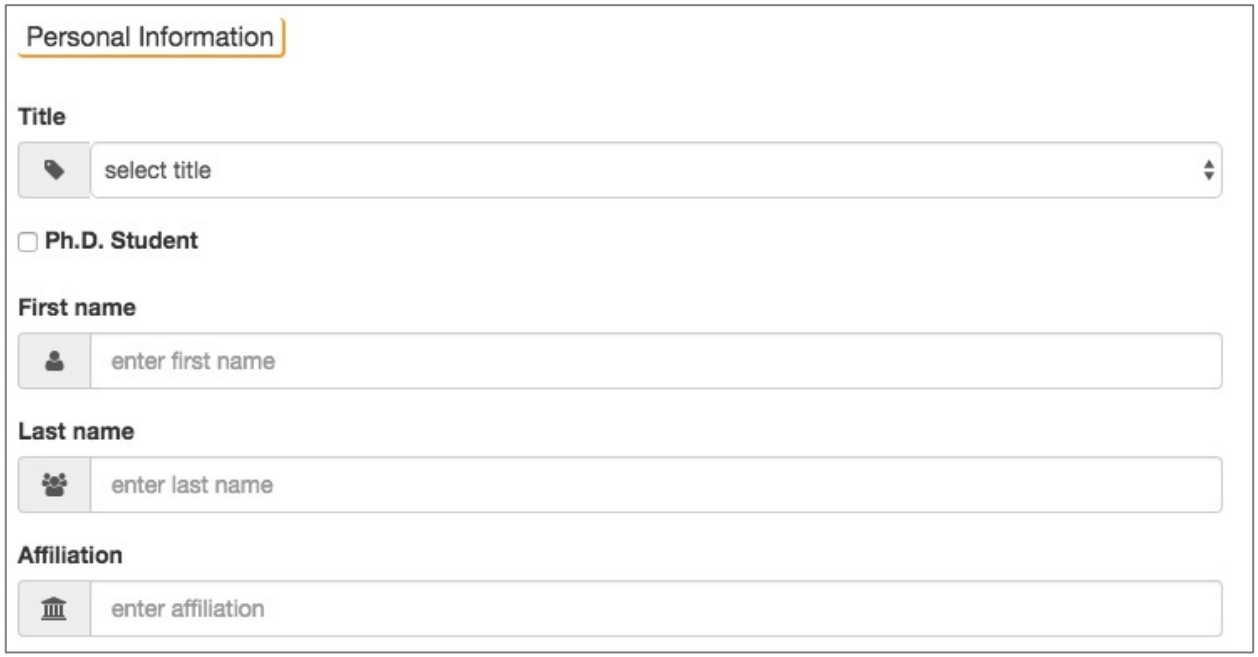
Es: Value must be less than or equal to 300

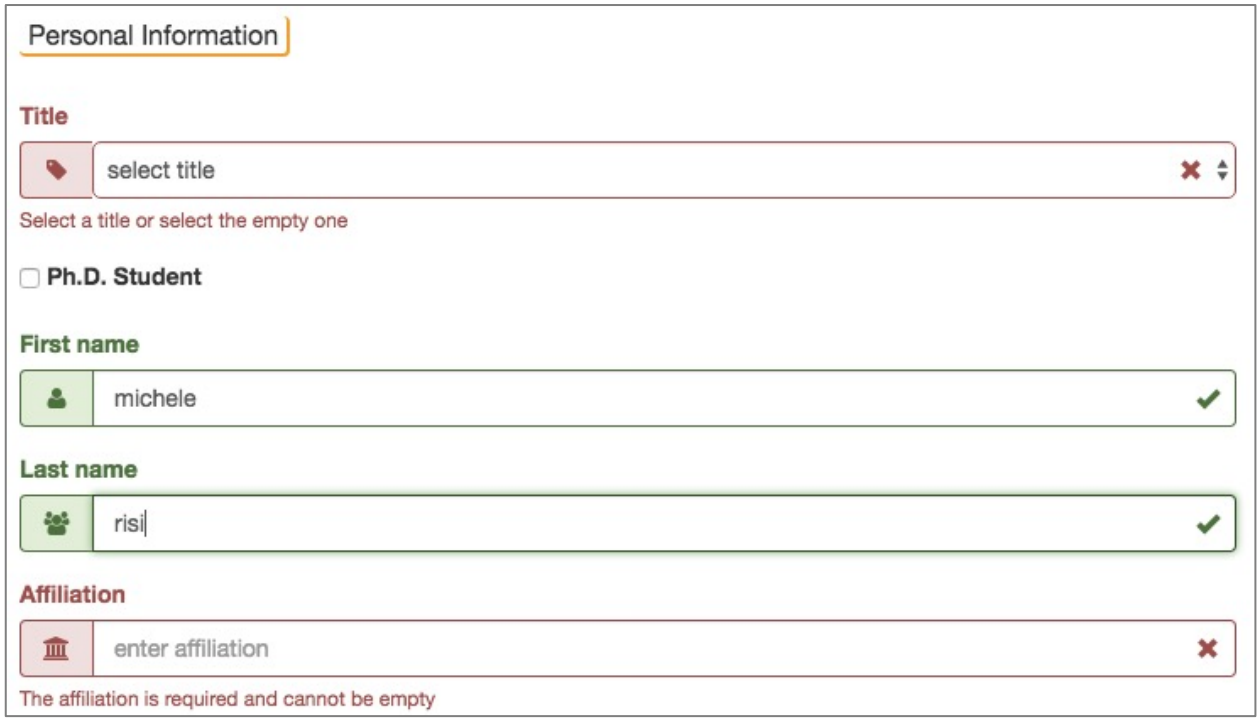
Manual validation

```
function ValidateEmail(uemail)
{

    var x = uemail.value;
    var atpos = x.indexOf("@");
    var dotpos = x.lastIndexOf(".");
    if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length) {
        alert("Not a valid e-mail address");
        uemail.focus();
        return false;
    }
    else
{
return true;
}
```

JavaScript can validate data

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

<h1>JavaScript Can Validate Input</h1>
<p>Please input a number between 1 and 10:</p>

<input id="numb">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>

<script>
function myFunction() {
    var x, text;

    // Get the value of the input field with id="numb"
    x = document.getElementById("numb").value;

    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```





Validation of a form

JavaScript errors - throw and try…catch

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

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

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

</body>
</html>
```

Can't find variable: adddlert

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>
```

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;
```

Chrome Developer Tools: dalla scheda Sources (nella modalità Inspector) inserire i breakpoint sul codice JavaScript

JavaScript declarations are hoisted

- In JavaScript, a variable can be declared after it has been used
- In other words; a variable can be used before it has been declared

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                     // Display x in the element

var x; // Declare x
```

Performance

```
for (i = 0; i < arr.length; i++) {
```

```
l = arr.length;
for (i = 0; i < l; i++) {
```

```
obj = document.getElementById("demo");
obj.innerHTML = "Hello";
```

```
var fullName = firstName + " " + lastName;
document.getElementById("demo").innerHTML = fullName;
```

```
document.getElementById("demo").innerHTML = firstName + " " + lastName
```

Best practice: delay JavaScript loading

- If possible, you can add your script to the page by code, after the page has loaded:

```
<script>
window.onload = downScripts;

function downScripts() {
    var element = document.createElement("script");
    element.src = "myScript.js";
    document.body.appendChild(element);
}
</script>
```

jQuery: Cos'è?

- jQuery (**http://jquery.com**) è una libreria di classi e funzioni JavaScript che permette al programmatore di compiere in modo facile quanto segue:
  - navigazione nel documento HTML
  - gestione degli eventi
  - animazioni
  - funzionalità AJAX

jQuery: Perché è utile?

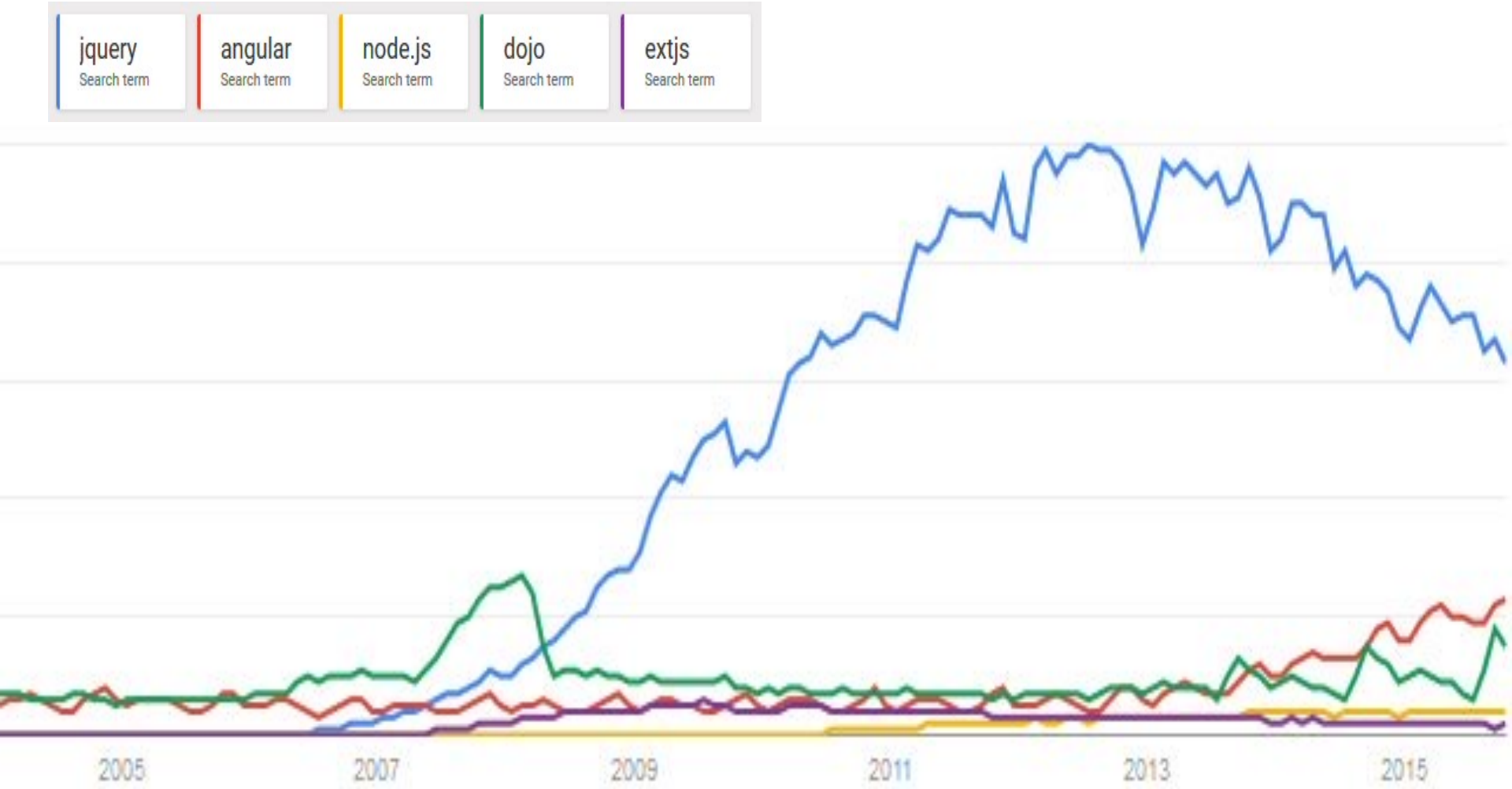
- Consente di scrivere codice in modo più compatto e ad alto livello

- Funge da "normalizzatore", fornendo dei costrutti conformi agli standard del W3C, indipendentemente dal browser su cui gira

JavaScript e jQuery: quali differenze?

- *Sebbene molti aspiranti sviluppatori Web si chiedano quale sia la differenza tra JavaScript e jQuery, in realtà* ***è bene sapere che sono la stessa cosa!!***
- Sinteticamente, jQuery è una libreria specifica per codice JavaScript (sviluppata da terzi) pensata appositamente per *semplificare l'attraversamento del DOM di una pagine HTML, la sua animazione, la gestione di eventi, e interazioni AJAX*
- Prima di jQuery, gli sviluppatori provvedevano a creare il proprio *"framework JavaScript"*
  - ciò permetteva loro di lavorare su specifici bug senza perdere tempo nel debugging di feature comuni
  - ciò ha portato alla realizzazione da parte di gruppi di sviluppatori di librerie open source e gratuite

Industry usage (Google searches)



Download

- Scaricare la libreria da

  https://jquery.com/download/

- Due release:
  - **compressa** (per siti in “produzione”)

    https://code.jquery.com/jquery-3.6.0.min.js
  - **non compressa** (per sviluppo)

    https://code.jquery.com/jquery-3.6.0.js

Loading jQuery: typical approach

```
...
<head>
<title>your title</title>
<link rel="stylesheet" href="css/your-styles.css"/>

<script src="scripts/jquery.js"></script>
<script src="scripts/your-script.js"></script>
</head>
...
```

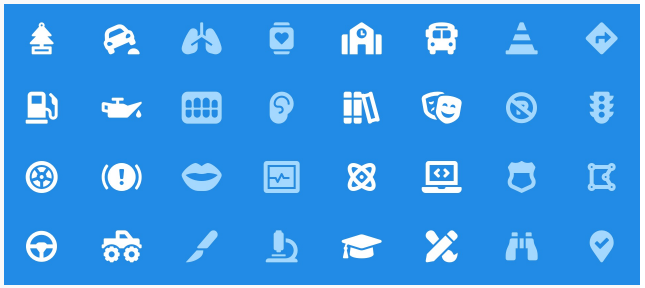
- You should load jQuery before loading your own scripts that make use of jQuery
- You should rename jquery-x.y.z.js to jquery.js

- **External import:**

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

...
```

Loading jQuery: modern approach

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Icon set core CSS -->
    <link href="css/fontawesome.min.css" rel="stylesheet">

    <!-- Bootstrap core CSS -->
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles -->
    <link href="css/template.css" rel="stylesheet">

    <title>Title</title>
  </head>
  <body>
    <!-- Begin page content -->
    <main role="main" class="container">
        <!-- ... -->
    </main>

    <!-- Bootstrap core JavaScript
    ================================================== -->
    <!-- Placed at the end of the document so the pages load faster -->
    <script src="js/jquery-3.4.1.min.js"></script>
    <script src="js/bootstrap.min.js"></script>
    <script src="js/template.js"></script>
    <script>
        // Custom scripts
    </script>
  </body>
</html>
```

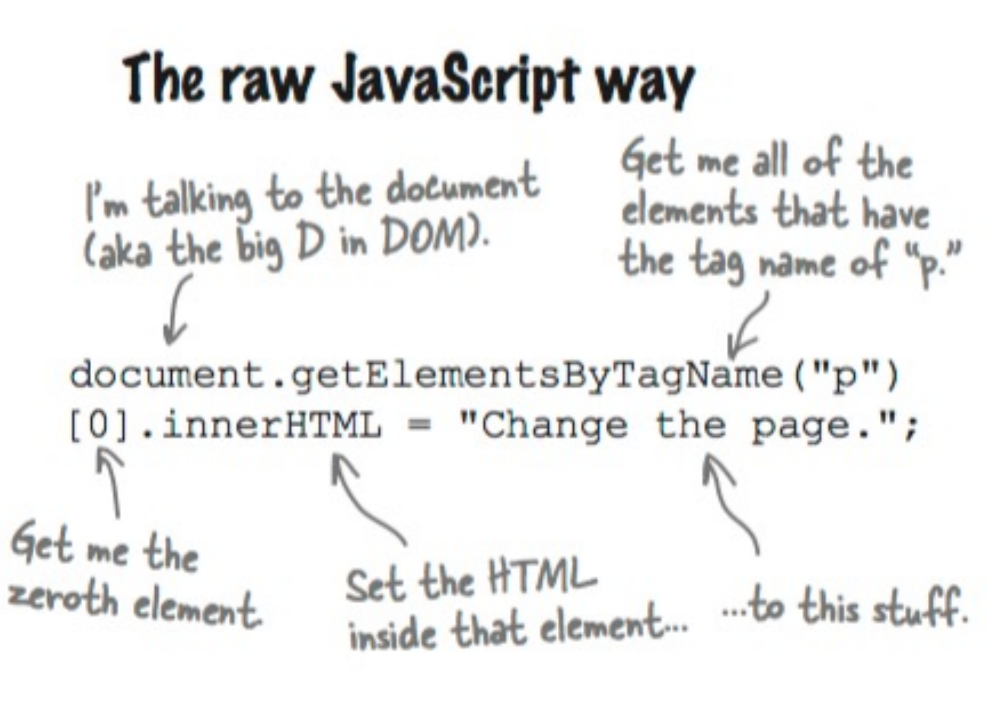


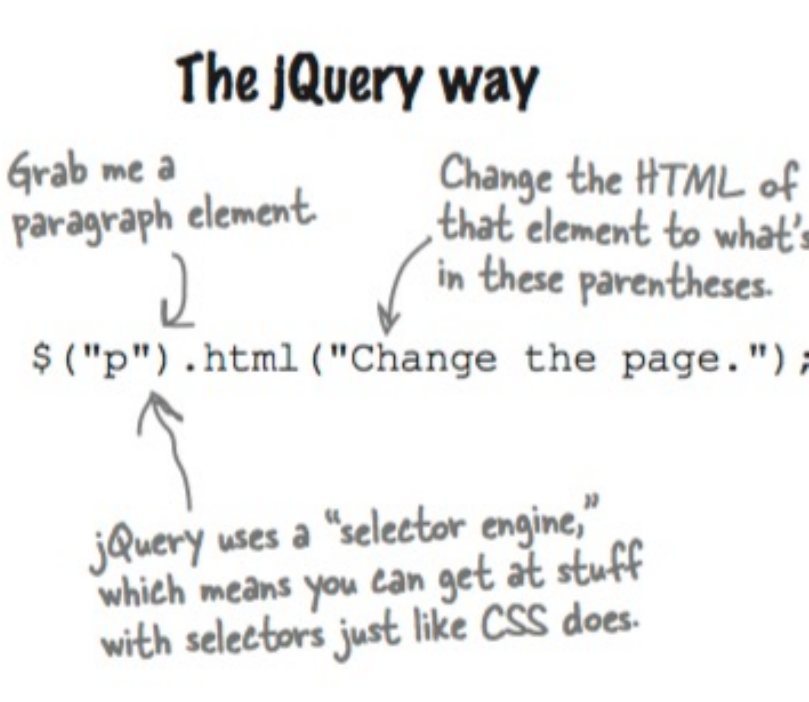
```
<i class="fa fa-camera-retro"></i> fa-camera-retro
```

fa-camera-retro

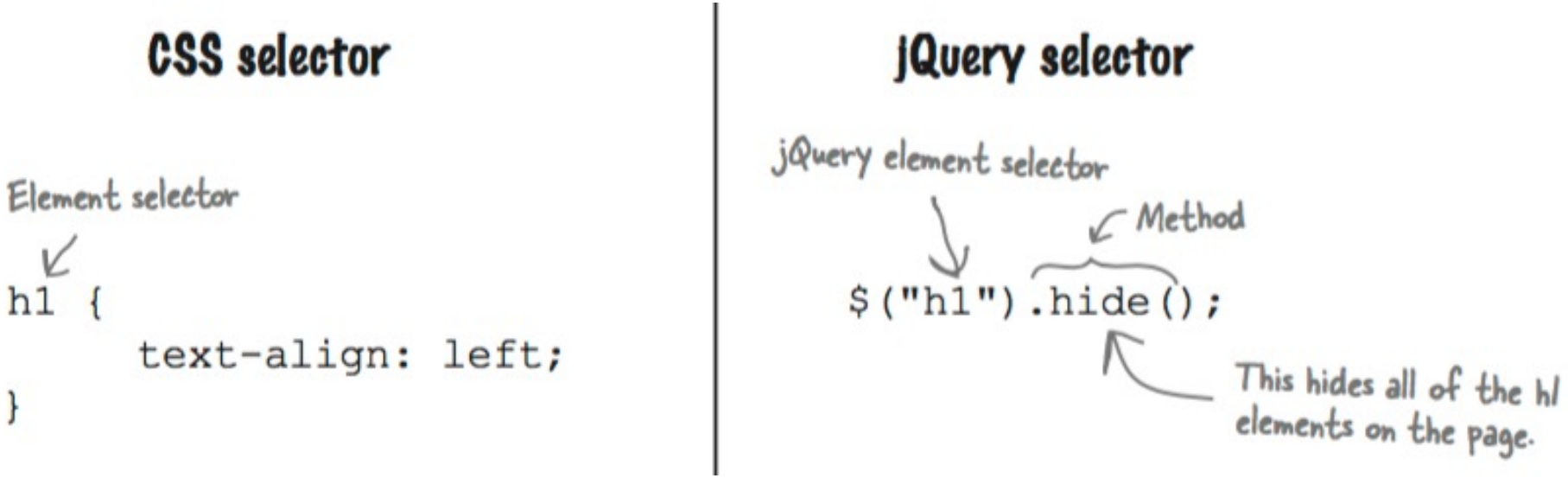
jQuery introduction

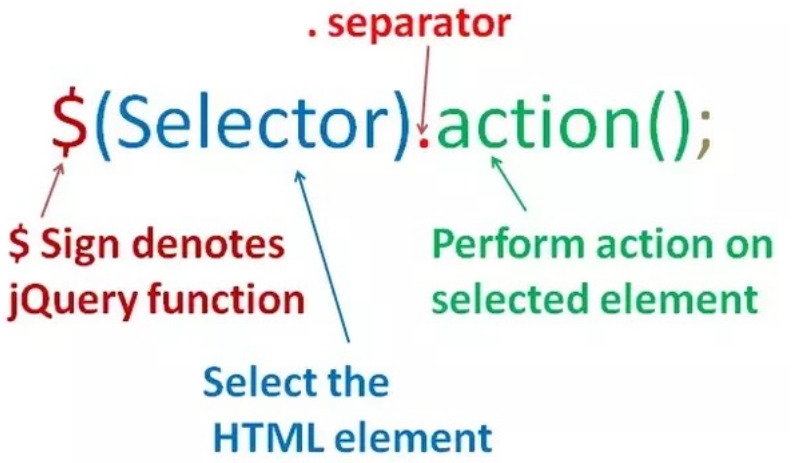
- jQuery *is* JavaScript, but a much more approachable version. When you want to control the DOM, jQuery makes it much easier





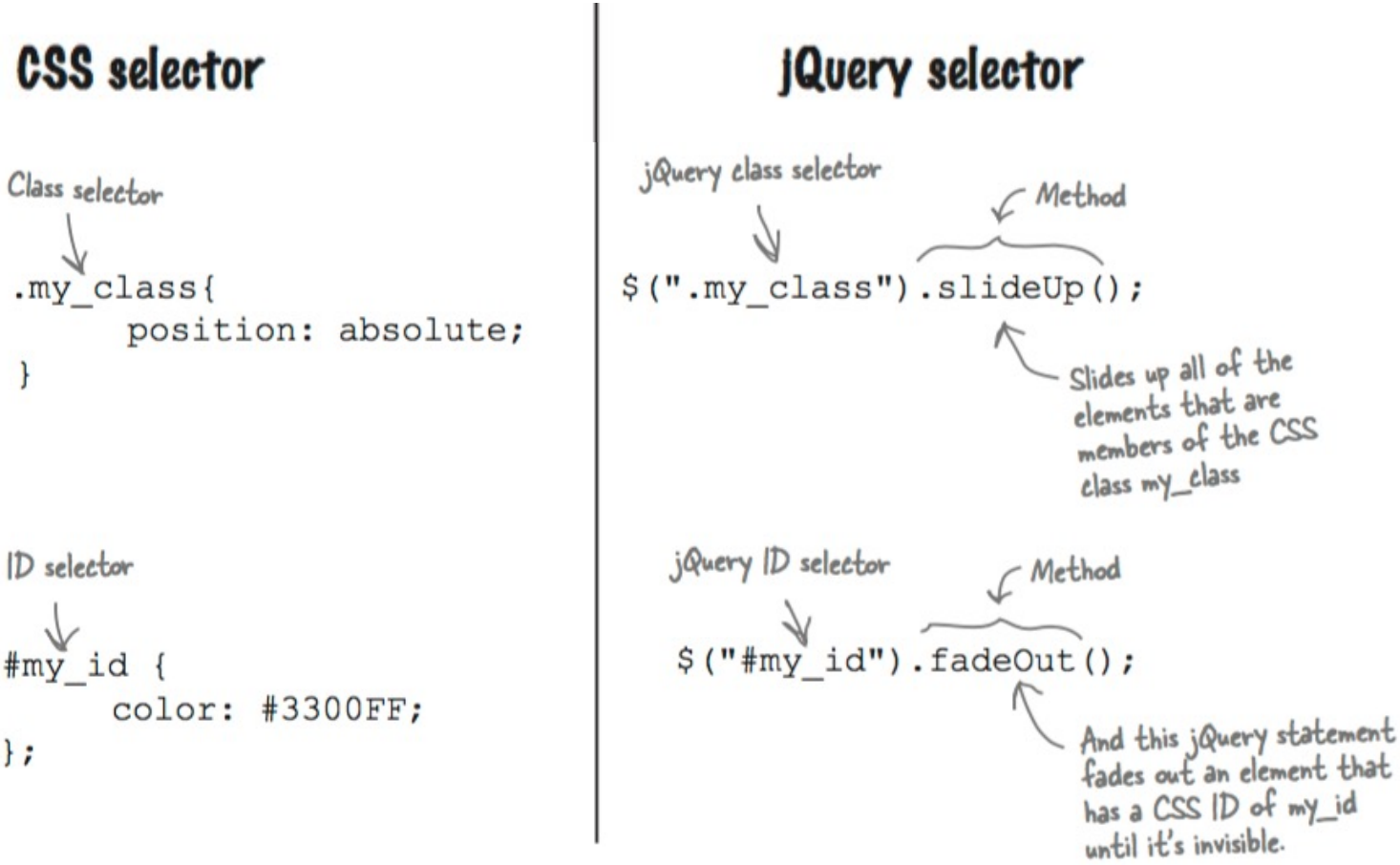
jQuery selects elements the same way CSS does





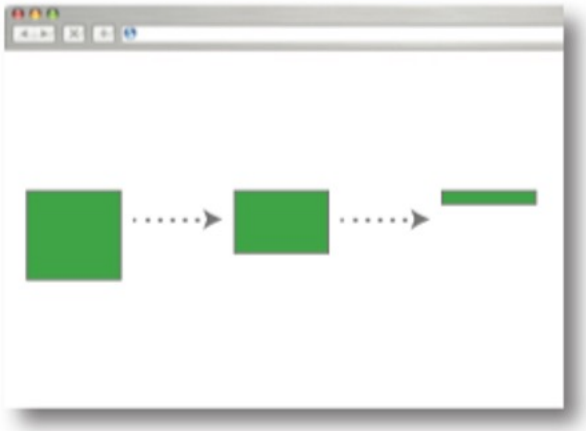
id & class

- CSS selectors select elements to add style to those elements
- jQuery selectors select elements to add behavior to those elements



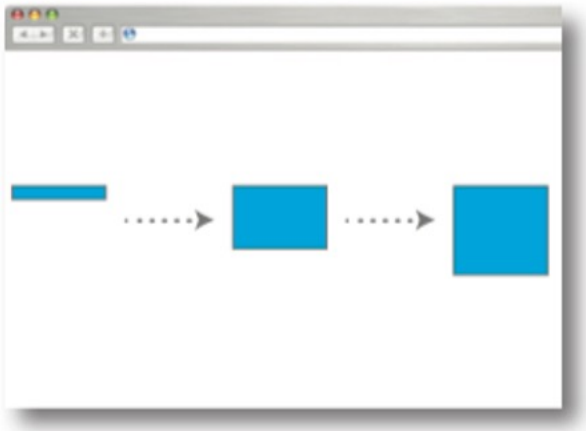
Slide on in...

```
$("div").slideUp();
```



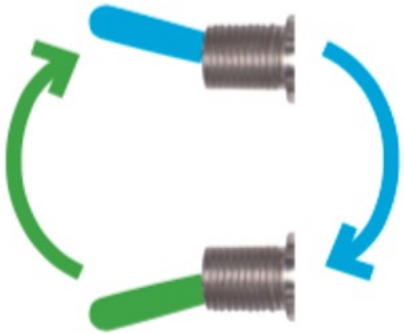
The slideUp method changes the height property of the element until it's 0, and then hides the element.

```
$("div").slideDown();
```



The slideDown method changes the height property of the element from 0 to whatever it's set to in the CSS style.

```
$("div").slideToggle();
```



The slideToggle action says, "If it's up, slide it down; if it's down, slide it up."

Fade

- The image gradually appears, going from invisible to fully visible

Here's what we want to fade in; in this case, it is an image.

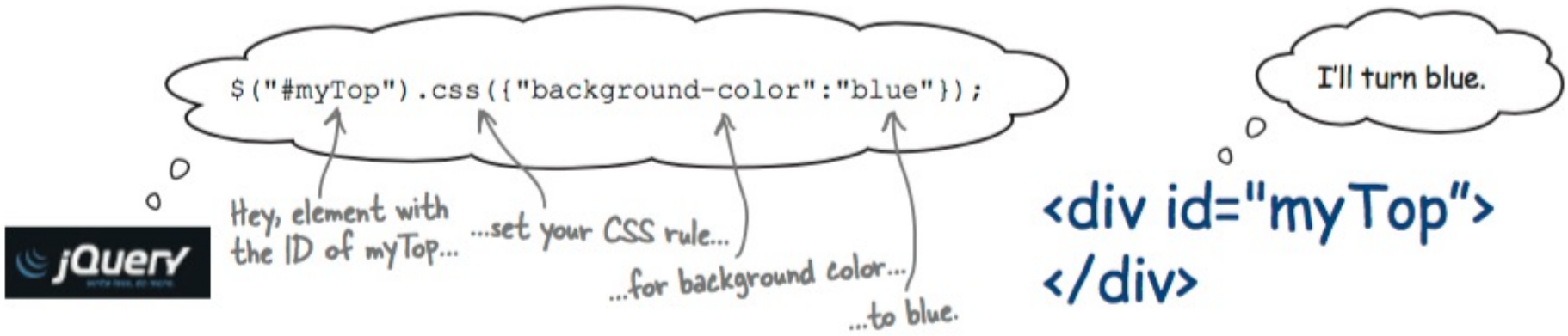
You can specify how fast it fades in by putting a value inside the parentheses, typically represented in milliseconds (ms).

```
$("img").fadeIn();
```

When an element fades in, it goes from being invisible (transparent) to being visible (opaque).

Other methods: fadeOut, fadeToggle

Change CSS rules




```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#myTop").css({"background-color": "blue", "color" : "white"});
});
</script>
</head>
<body>

<h1 id="myTop">This is a text</h1>

</body>
</html>
```

This is a text

$(this) keyword: select the current HTML element

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js">
</script>
<script>
$(document).ready(function(){
        $(".guess_box").click( function() {
            var discount = Math.floor((Math.random()*5) + 5);
            var discount_msg = "<p>Your Discount is "+discount+"%</p>";
            $(this).append(discount_msg);
        });
});
</script>
</head>
<body>

<p class="guess_box">...</p>


</body>
</html>
```

...

Your Discount is 6%

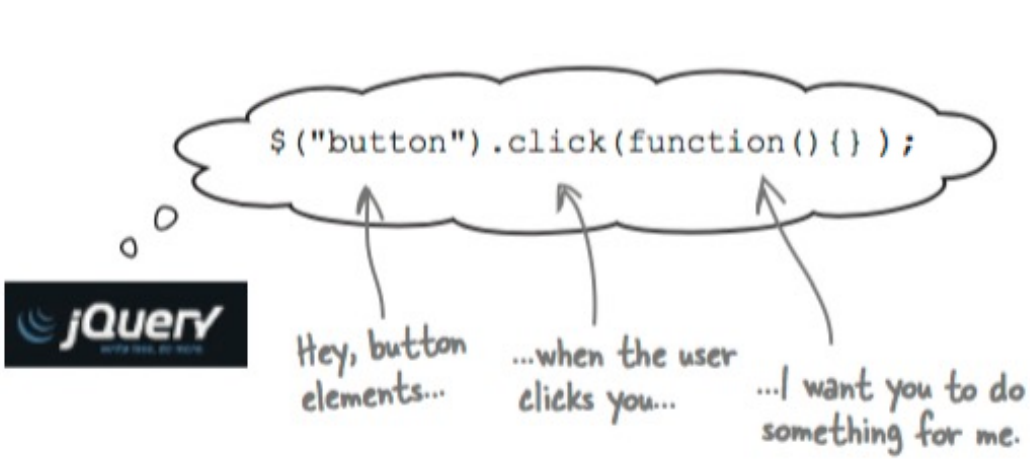
...

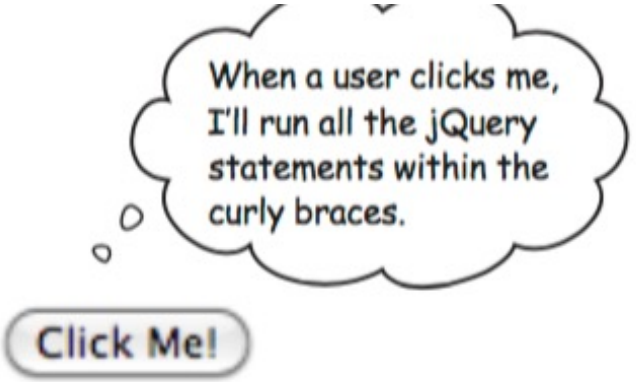
Your Discount is 6%

Your Discount is 6%

Events and event handlers

- An event represents the precise moment when something happens
- Examples:
  - moving a mouse over an element
  - selecting a radio button
  - clicking on an element





Events and event handlers (2)

- **click()**
  - The click() method attaches an event handler function to an HTML element
  - The function is executed when the user clicks on the HTML element
  - The following example says: When a click event fires on a <p> element; hide the current <p> element:

```
$("p").click(function(){
    $(this).hide();
});
```

```
$("#hide").click(function(){
    $("p").hide();
});

$("#show").click(function(){
    $("p").show();
});

$("button").click(function(){
    $("p").toggle();
});
```

Events and event handlers (3)

- **dblclick()**
  - The dblclick() method attaches an event handler function to an HTML element
  - The function is executed when the user double-clicks on the HTML element:

```
$("p").dblclick(function(){
    $(this).hide();
});
```

Events and event handlers (4)

- **mouseenter()**
  - The mouseenter() method attaches an event handler function to an HTML element
  - The function is executed when the mouse pointer enters the HTML element:

```
$("#p1").mouseenter(function(){
    alert("You entered p1!");
});
```

- **mouseleave()**
  - The mouseleave() method attaches an event handler function to an HTML element
  - The function is executed when the mouse pointer leaves the HTML element

Events and event handlers (5)

- **mousedown()**
  - The mousedown() method attaches an event handler function to an HTML element
  - The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element
- **mouseup()**
  - The mouseup() method attaches an event handler function to an HTML element
  - The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element

Events and event handlers (6)

- **hover()**
  - The hover() method takes two functions and is a combination of the mouseenter() and mouseleave() methods
  - The first function is executed when the mouse enters the HTML element, and the second function is executed when the mouse leaves the HTML element:

```
$("#p1").hover(function().{
                    alert("You entered p1!");
                    },
                  function() {
                    alert("Bye! You now leave p1!");
                    }
);
```

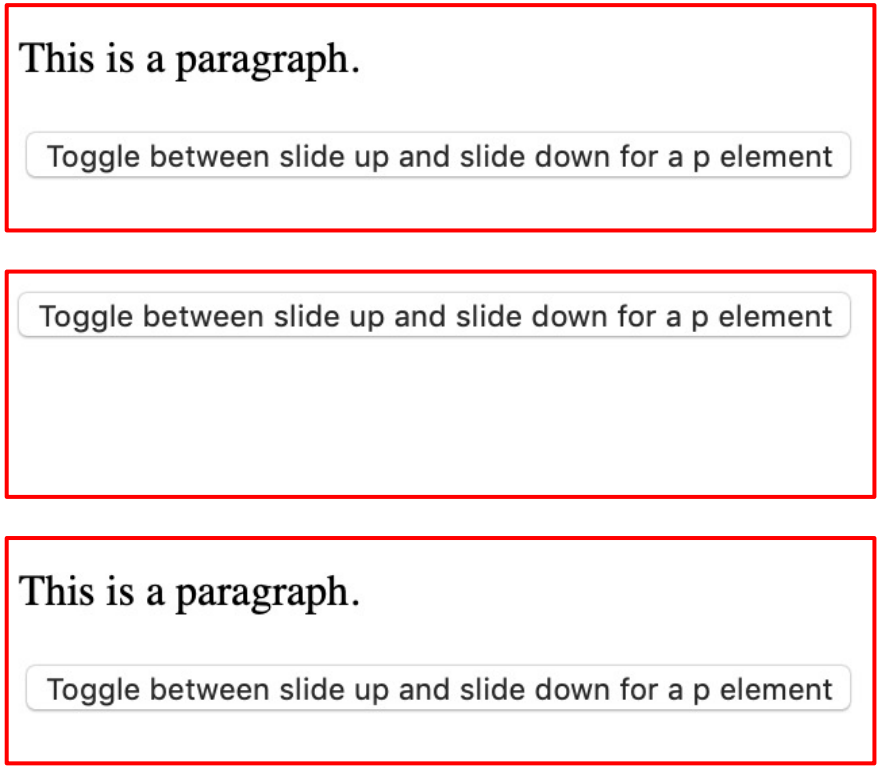
Special Events

- The **$(document).ready()** method allows us to execute a function when the document is fully loaded:

```
$(document).ready(function(){

    // jQuery methods go here...

});
```

Example: document.ready and click

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").slideToggle();
    });
});
</script>
</head>
<body>

<p>This is a paragraph.</p>

<button>Toggle between slide up and slide down for a p element</button>

</body>
</html>
```



Manipulating DOM Elements: commonly used functions

- $("#some-id").val()
  - Returns value of input element. Used on 1-element sets
- $("selector").each(function() { })
  - Calls function on each element. “This” set to element
- $("selector").addclass("name")
  - Adds CSS class name to each. Also removeclass, toggleclass
- $("selector").hide()
  - Makes invisible (display: none). Also show, fadeout, fadein, etc.
- $("selector").click(function() { })
  - Adds onclick handler. Also change, focus, mouseover, etc.
- $("selector").html("<tag>some html</tag>")
  - Sets the innerHtml of each element. Also append, prepend

Riferimenti

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

- jQuery:
  - **https://api.jquery.com**