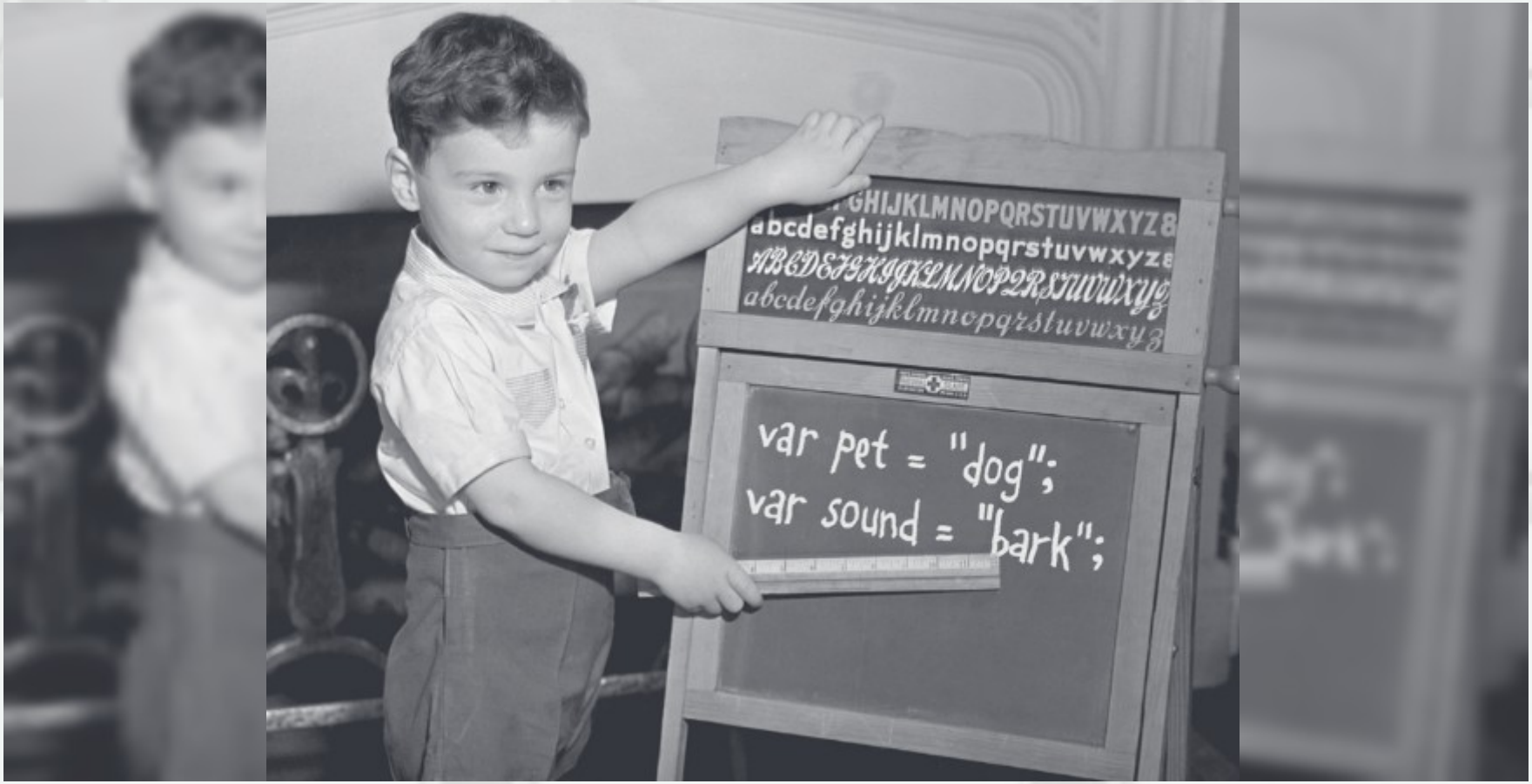
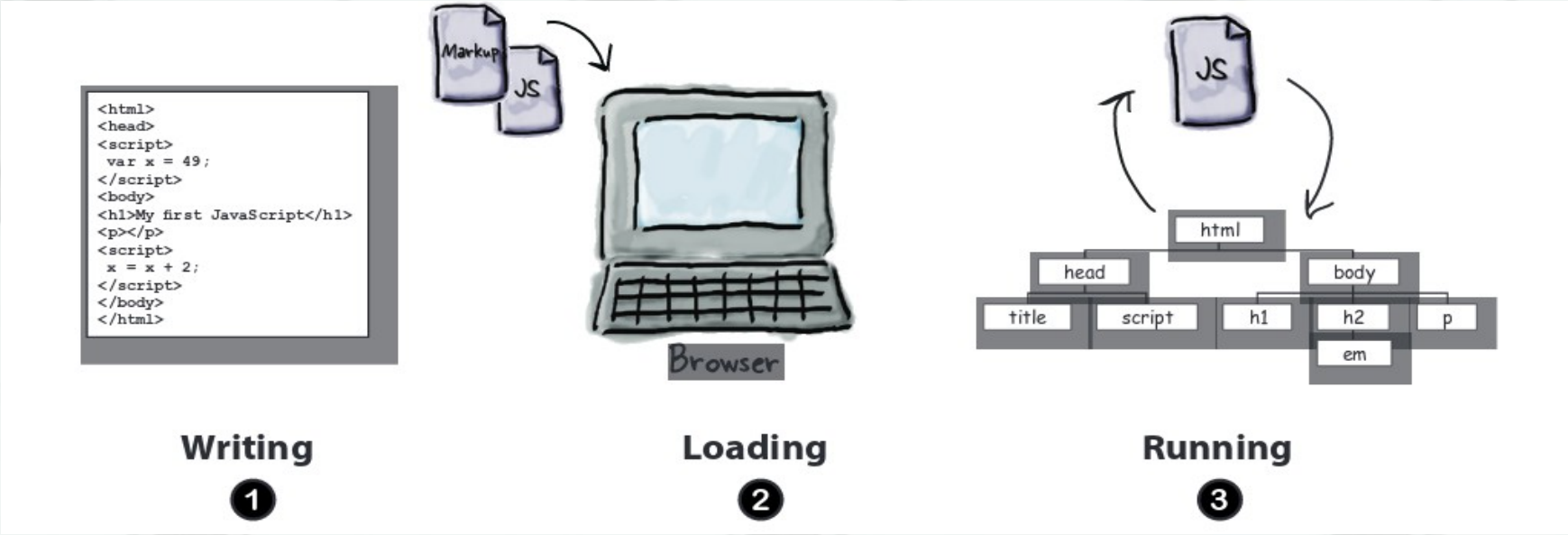
HTML5
CAPITOLO 2

Rusnac Dan

Introduzione a JavaScript e DOM (Document Object Model)



Uso e funzionalità di JavaScript



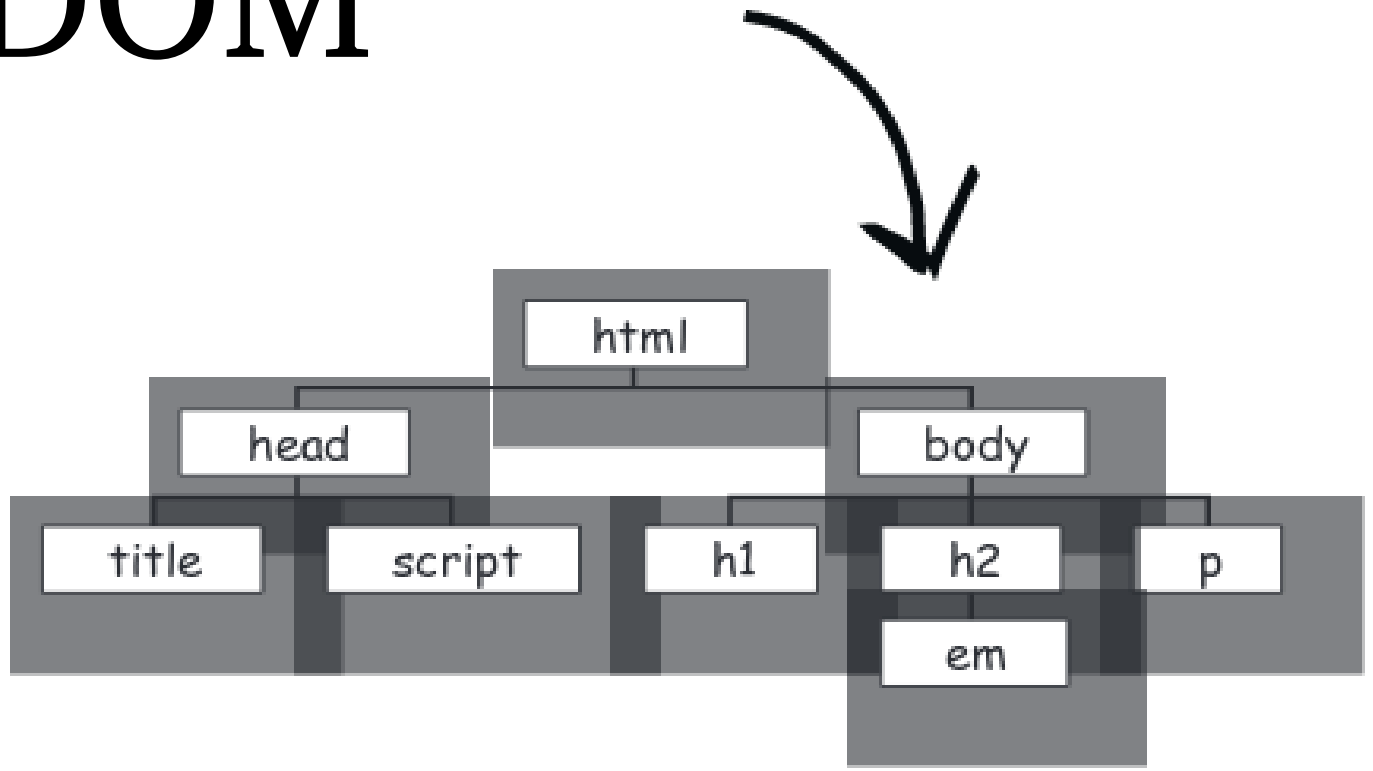

```
<html>
<head>
<script>
 var x = 49;
</script>
<body>
<h1>My first JavaScript</h1>
<p></p>
<script>
 x = x + 2;
</script>
</body>
</html>
```

Creazione e **salvataggio** dei codici html e JavaScript

Il browser **carica** la pagina e **analizza** i suoi contenuti.
Quando trova il codice JavaScript questo viene "**corretto**" ed **eseguito**.
Viene poi creato il **DOM.**

JavaScript **resta in esecuzione** e usa il DOM per controllorare, cambiare o aggiornare la pagina, chiedere al browser altri dati...

Esempio DOM



Cosa possiamo fare con JavaScript?

1 make a statement

Create a variable and assign values, add things together, calculate things, use built-in functionality from a JavaScript library.

```
var temp = 98.6;
var beanCounter = 4;
var reallyCool = true;
var motto = "I Rule";
temp = (temp - 32) * 5 / 9;
motto = motto + " and so do you!";
var pos = Math.random();
```

❷ do things more than once, or twice

Perform statements over and over, as many times as you need to.

```
while (beanCounter > 0) {
   processBeans();
   beanCounter = beanCounter - 1;
}
```

❸ make decisions

Write code that is conditional, depending on the state of your app.

```
if (isReallyCool) {
   invite = "You're invited!";
}  else {
   invite = "Sorry, we're at capacity.";
}
```

Declaring a variable

Variables hold things. With JavaScript they can hold lots of different things. Let's declare a few variables that hold things:

```
var winners = 2;
var boilingPt = 212.0;
var name = "Dr. Evil";
var isEligible = false;
```

Integer numeric values.

Or floating point numeric values.

Or, strings of characters (we call those "strings," for short).

Or a boolean value, which is true or false.

JavaSccript non tiene conto della tipologia della variabile, quindi qualsiasi variabile può contenere numeri, stringhe o boolean.

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

Avoid these as variable names!

Avoid these as variable names!

Making more decisions... and, adding a catchall

You can provide a catchall for your `if` statements as well—a final `else` that is run if all the other conditions fail. Let's add a few more if/elses and also a catchall:

Notice we changed this to only happen when scoops is precisely 3.

```
if (scoops == 3) {
   alert("Ice cream is running low!");
} else if (scoops > 9) {
   alert("Eat faster, the ice cream is going to melt!");
} else if (scoops == 2) {
   alert("Going once!");
} else if (scoops == 1) {
   alert("Going twice!");
} else if (scoops == 0) {
   alert("Gone!");
} else {
   alert("Still lots of ice cream left, come and get it.");
}
```

We've added additional conditions to have a countdown to zero scoops.

Here's our catchall; if none of the conditions above are true, then this block is guaranteed to execute.


Everything seems to work well if I add numbers to numbers or strings to strings, but what if I add a number to a string? Or an integer to a floating point number?

In JavaScript è possibile fare operazioni tra tipi di variabili diverse.
Ad esempio:

```
message = 2 + ”ciao” ;
```

JavaScript capisce che la stringa “ciao” non potrà mai essere un numero, dunque tratta l'espressione come un'operazione tra stringhe, convertendo il 2 in stringa.

```
Message = 2 * 3.1 ;
```

JavaScript converte il 2 in un float, dando come resultato 6,2.

Vediamo adesso come inserire il nostro JavaScript all'interno delle pagine e come questo interagisce con i suoi elementi.

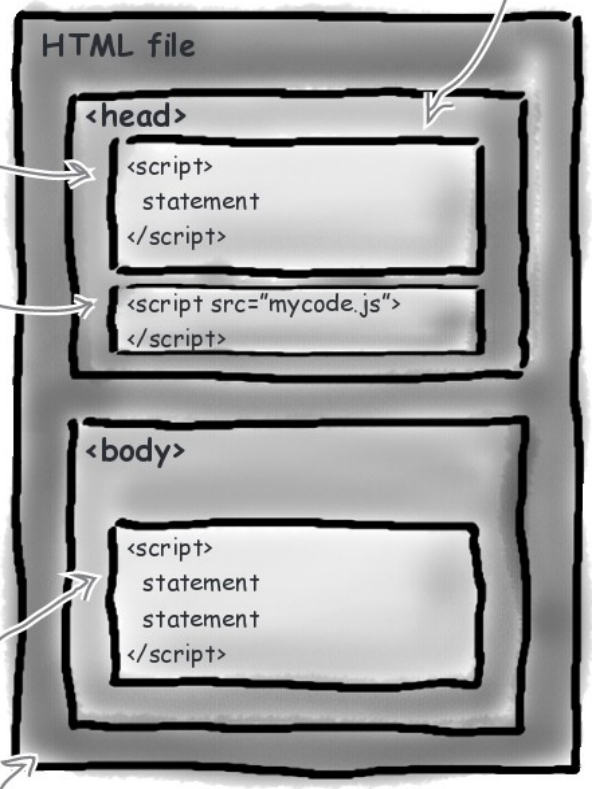
Dove e quando?

- Si inserisce un elemento di tipo <*script*> nell'**head** della pagina.
- Si può usare un **collegamento** ad un altro file contente il codice JavaScript.
  Basta inserire l'**URL** del file all'interno dell'attributo *scr* della tag di <*script*> (se sono nella stessa cartella basta scrivere il nome del file).
- Si aggiunge il codice all'interno del **body** della pagina.
  Lo si può tranquillamente inserire come nell'head oppure tramite un collegamento.

Place <script> elements in the <head> of your HTML to have them executed before the page loads.

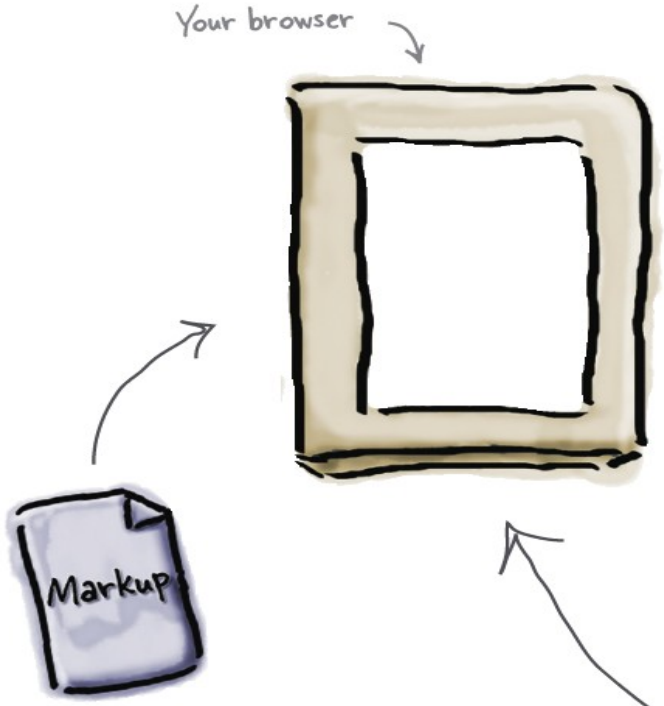
You can type your code right into your web page, or reference a separate JavaScript file using the src attribute of the script tag.

Or you can place your code (or a reference to your code) in the body. This code gets executed as the body is loaded.

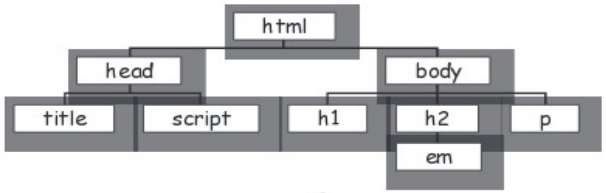


Most of the time code is added to the head of the page. There are some slight performance advantages to adding your code at the end of body, but only if you really need to super-optimize your page's performance.

Interazione con le pagine



Quando la pagina viene caricata, il browser crea il DOM, che contiene tutti gli elementi del tuo codice HTML.



Quando JavaScript modifica il DOM, il browser aggiorna la pagina.

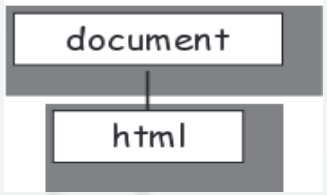
Il codice JavaScript può interagire con il DOM per accedere ad elementi al suo interno oppure per creare o elimanere altri elementi.

Come costruire il nostro DOM?

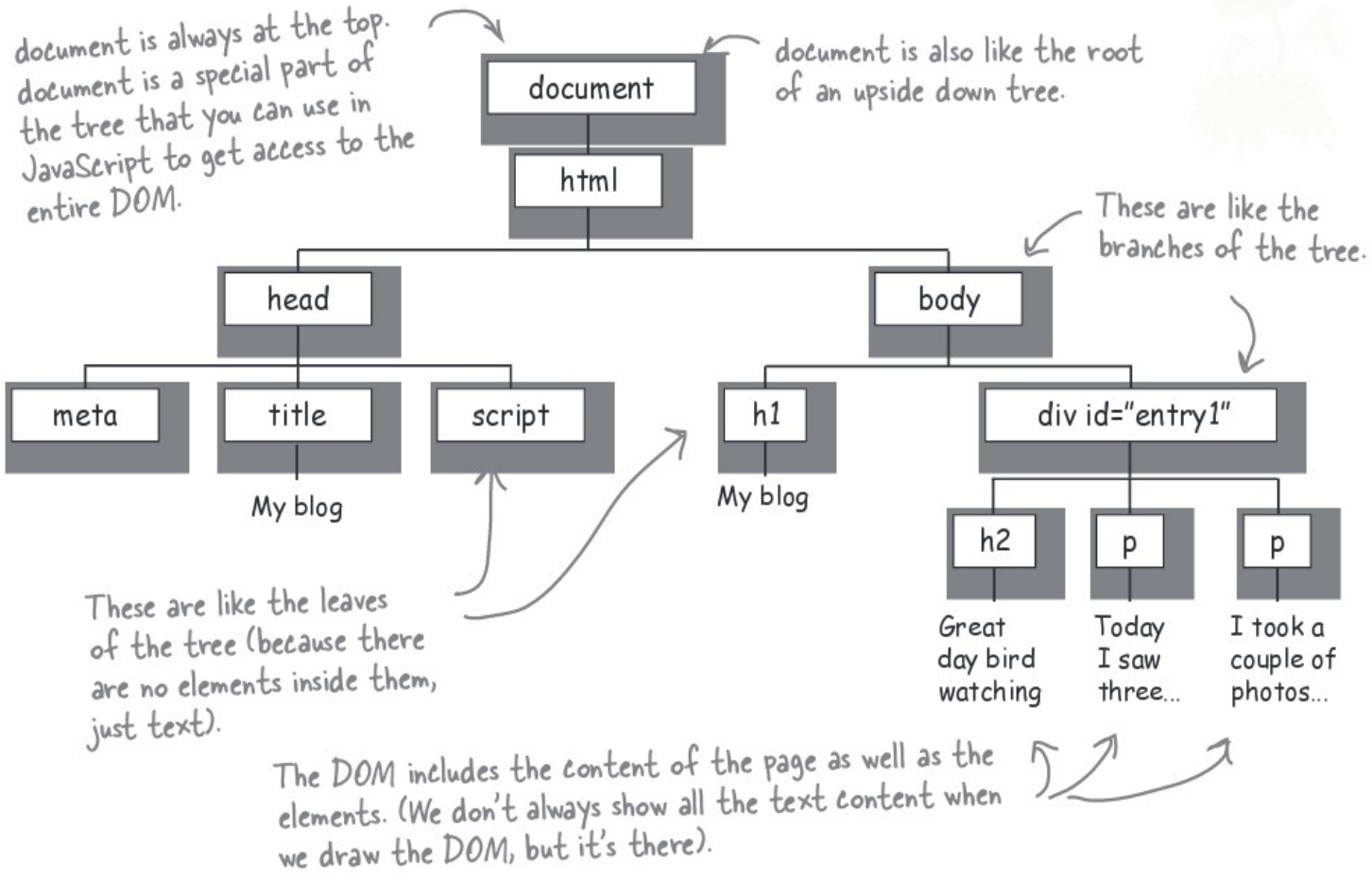
```
<!doctype html>
<html lang="en">
<head>
  <title>My blog</title>
  <meta charset="utf-8">
  <script src="blog.js"></script>
</head>
<body>
  <h1>My blog</h1>
  <div id="entry1">
    <h2>Great day bird watching</h2>
    <p>
      Today I saw three ducks!
      I named them
      Huey, Louie, and Dewey.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

Istruzioni:

- Si crea il primo nodo: **document**;
- Si prende l'elemento di “livello” più alto, nel nostro caso: **<html>;**



- Viene aggiunto al grafo (come figlio di tal elemento) ogni elemento che sta all'interno di un'altro.
- Si ripete l'ultimo procedimento fino a quando non finiscono gli elementi.


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).

Possiamo lavorare con il DOM solo dopo il caricamento della pagina!

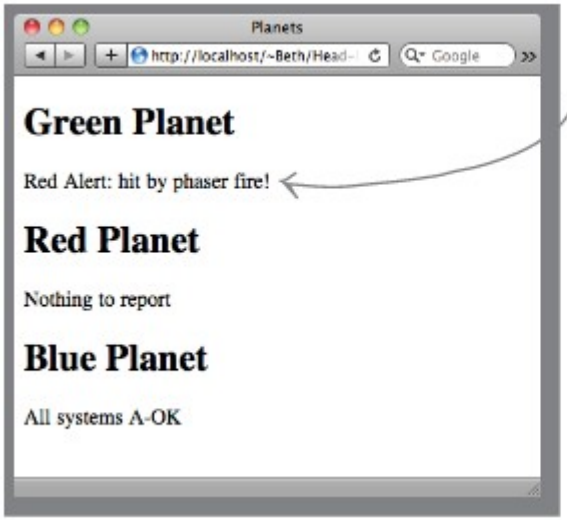
```
<script>
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
}

window.onload = init;
</script>
```

First, create a function named init and put your existing code in the function.

Notice that your code goes between an opening { and a closing }.

Here, we're setting the value of the window.onload property to the function name.

This says when the page is fully loaded, execute the code that is in init.



Yes! Now we see the new content in the green planet <p> element. Isn't it great?

Well, what IS great is that now you know how to tell the browser to wait until the DOM has completely loaded before running code that accesses elements.

Altri utilizzi del DOM:

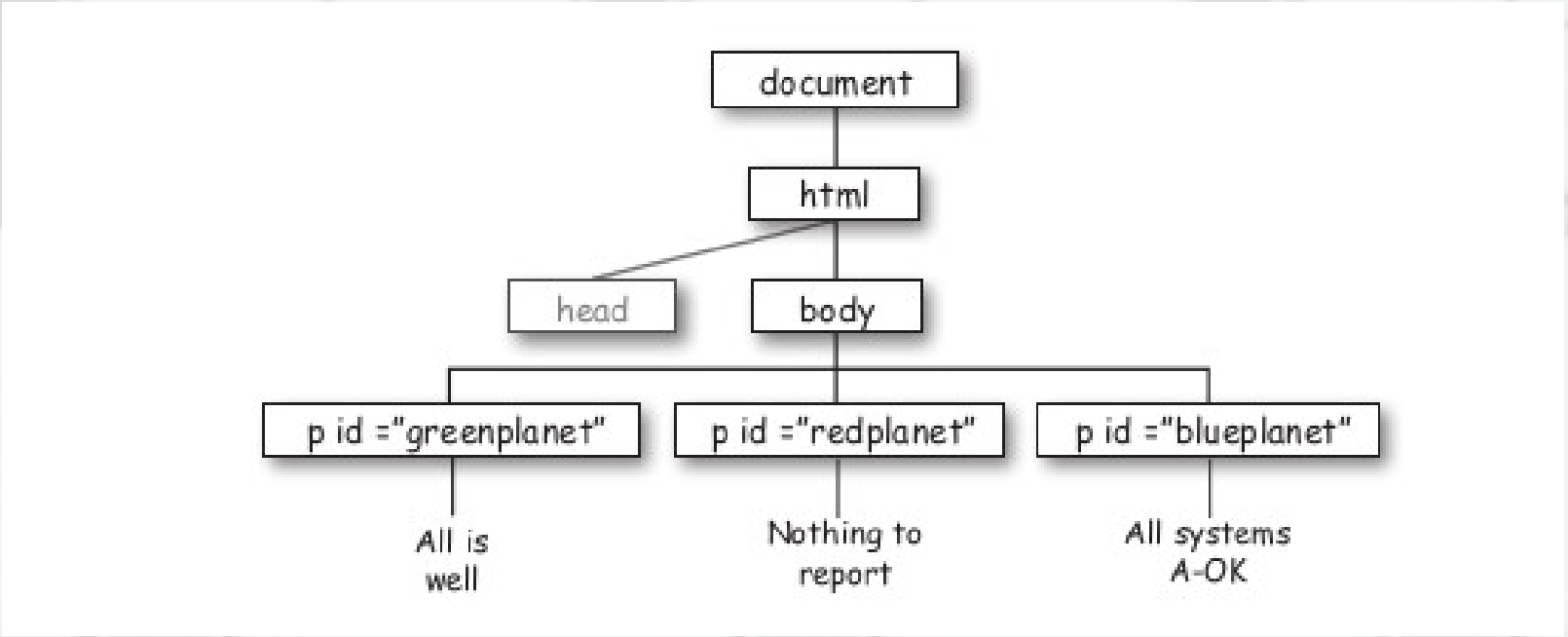
- Si possono prendere elementi dal suo interno (es: document.getElementById);

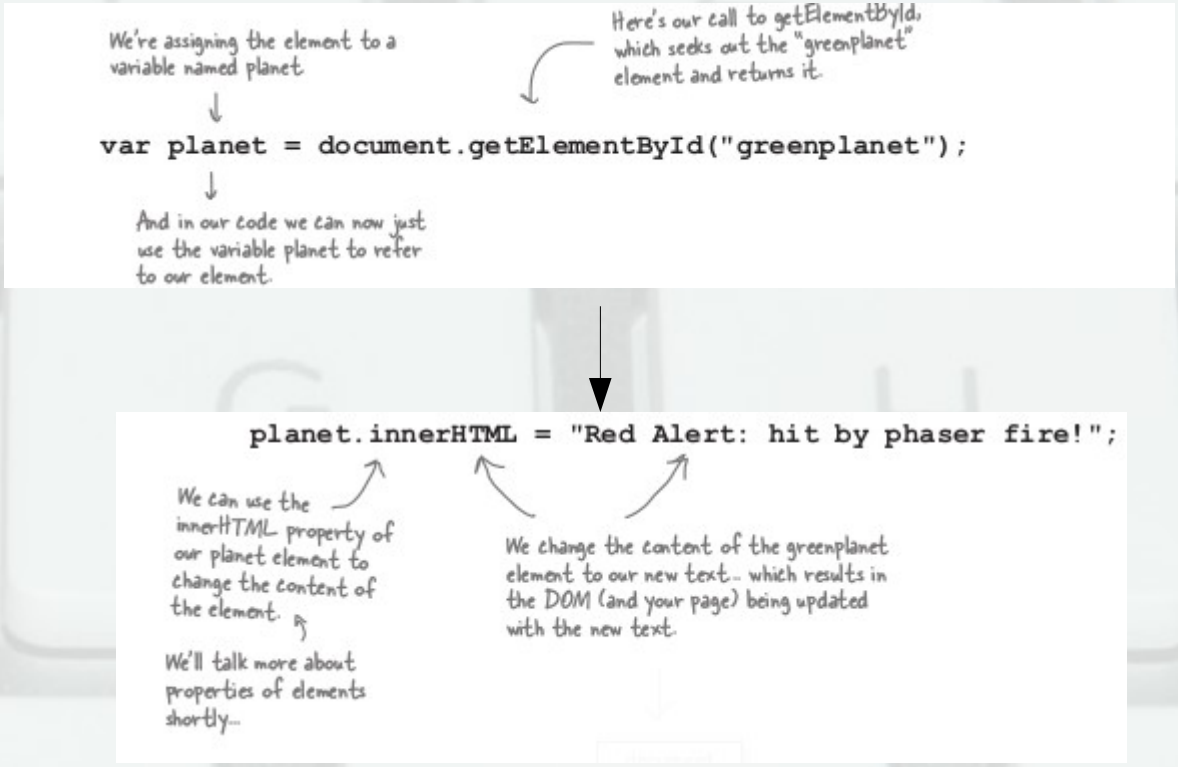
- Si possono creare e aggiungere elementi nuovi;

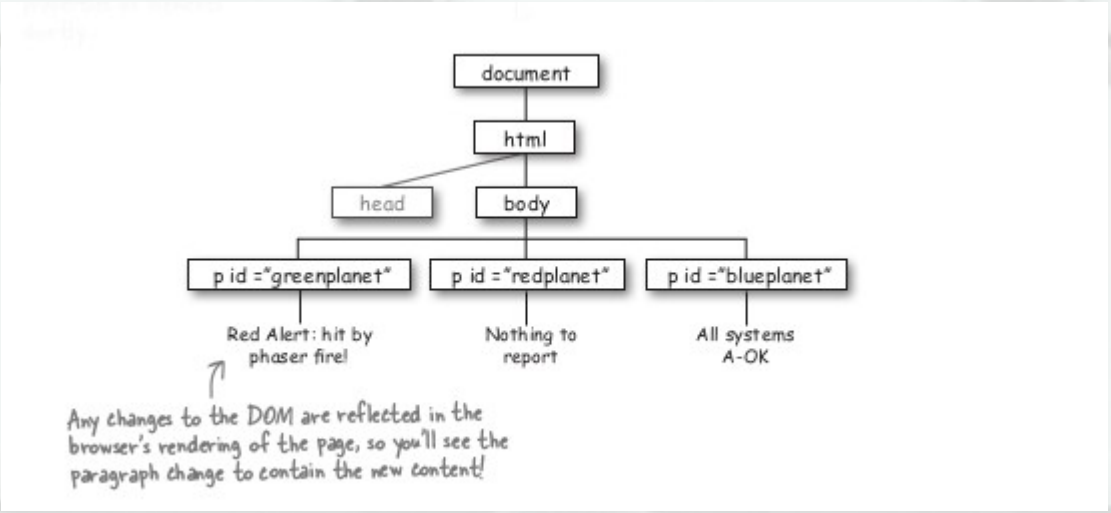
- Si possono rimuovere elementi;

- Si può accedere agli attributi degli elementi, con la possibilità di cambiare classe o altro.

Esempio


document
html
head
body
p id ="greenplanet"
p id ="redplanet"
p id ="blueplanet"
All is well
Nothing to report
All systems A-OK


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'll talk more about properties of elements shortly...
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.


document
html
head
body
p id ="greenplanet"
p id ="redplanet"
p id ="blueplanet"
Red Alert: hit by phaser fire!
Nothing to report
All systems A-OK
Any changes to the DOM are reflected in the browser's rendering of the page, so you'll see the paragraph change to contain the new content!


The End