

JavaScript (1)



Marina Ribaudó, marina.ribaudó@unige.it

Programmazione lato client

2

JavaScript is THE scripting language of the Web

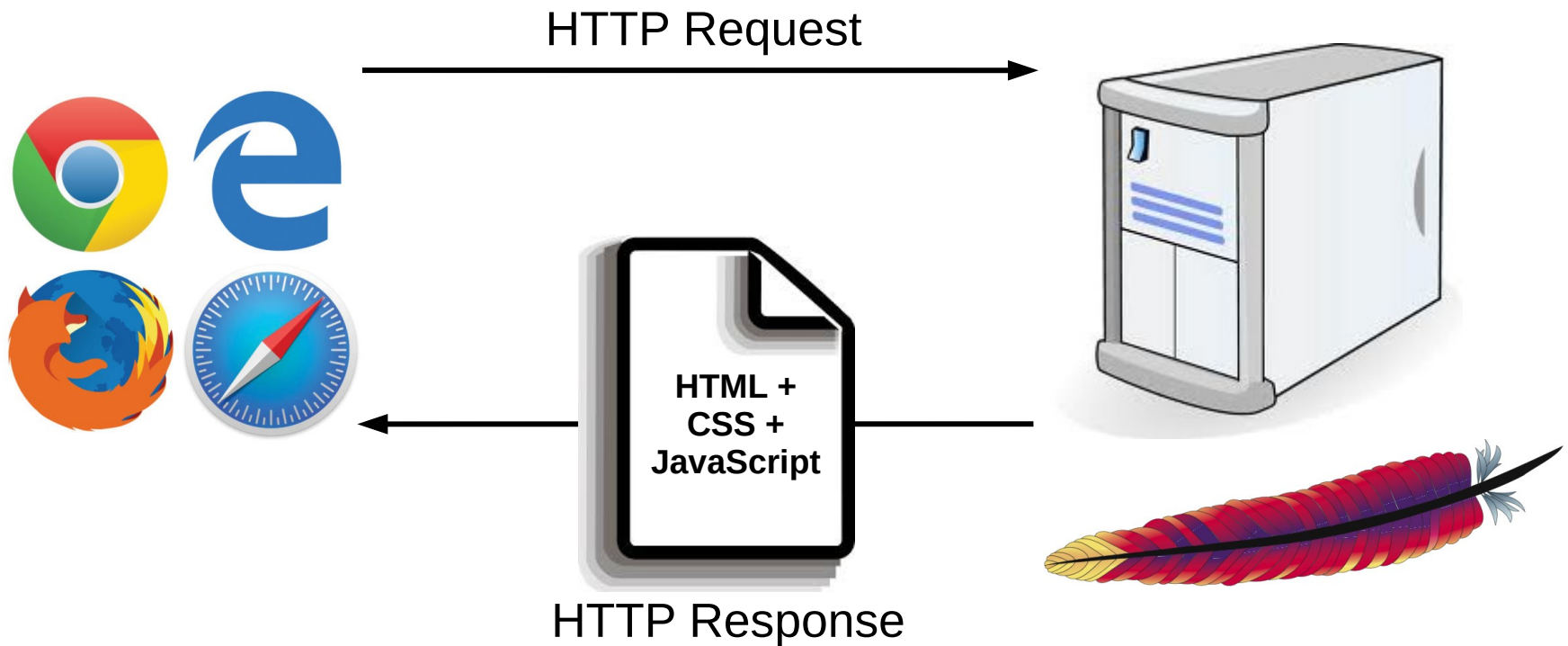


ECMAScript is the official name of the JavaScript standard
(European Computer Manufacture's Association)

<https://ecma-international.org/technical-committees/tc39/>

JavaScript

3



JavaScript

4

variabili
espressioni
istruzioni
funzioni
oggetti e array

core language
ECMAScript



client-side
DOM + BOM

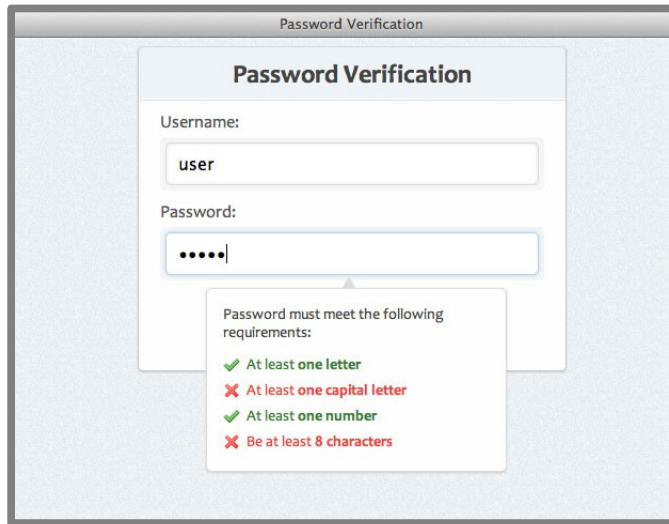
server-side

Per esempio Node.js

Usi di JavaScript

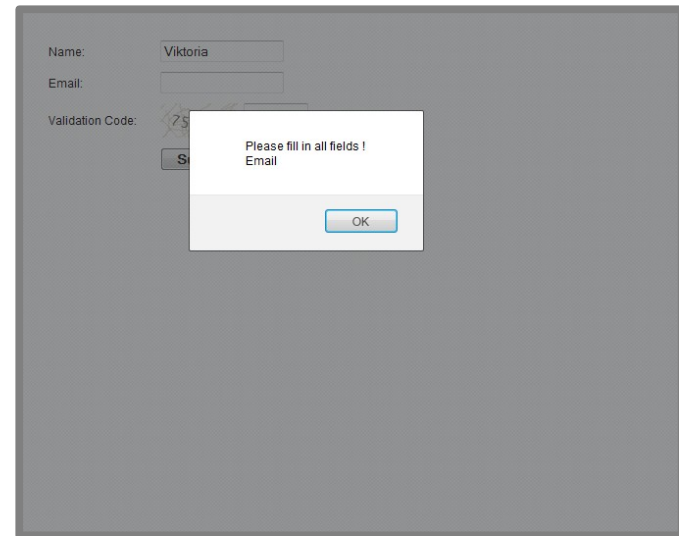
5

- Validazione input
- Apertura nuove finestre



A screenshot of a web form titled "Password Verification". It contains two input fields: "Username:" with the value "user" and "Password:" with masked characters ".....". Below the password field, a message box states: "Password must meet the following requirements:" followed by four items: "✓ At least one letter", "✗ At least one capital letter", "✓ At least one number", and "✗ Be at least 8 characters".

**NB: oggi in molti casi
usiamo HTML5**

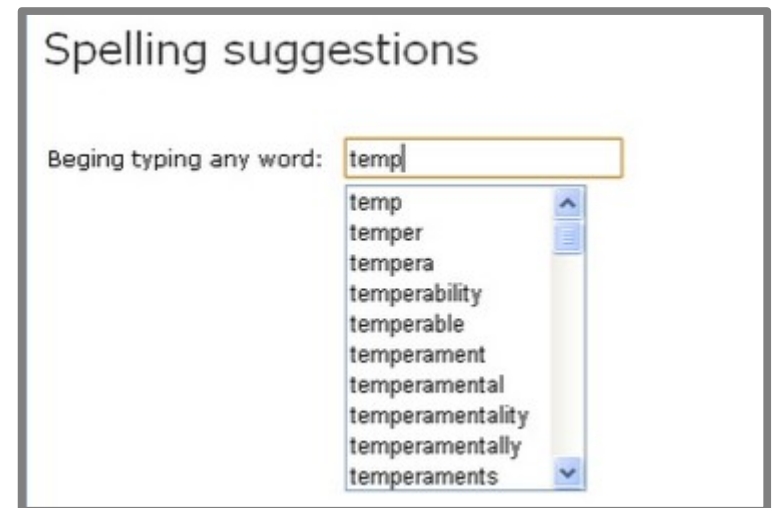
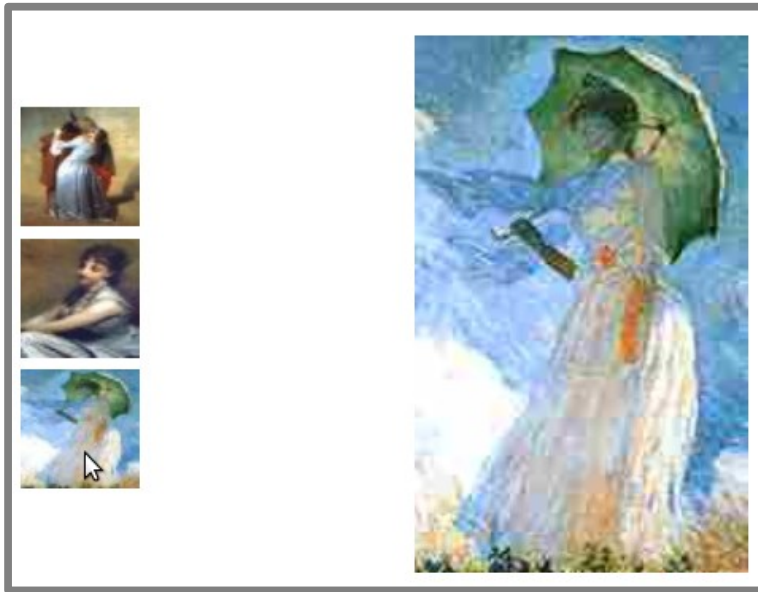


A screenshot of a registration form with fields for "Name:" (filled with "Viktoria"), "Email:", and "Validation Code:". A modal dialog box is displayed in the foreground with the text "Please fill in all fields !
Email" and an "OK" button.

Usi di JavaScript

6

- Rollover, Ajax, Fetch



NB: oggi usiamo i CSS

Dove si scrivono gli script?

7

- Gli script **sono caricati ed eseguiti dal browser** durante la lettura del codice HTML oppure in risposta al verificarsi di un evento
- **Attenzione:** uno script non può fare riferimento a elementi della pagina (link, immagini, form) che non siano ancora stati definiti

Esempio

```
<script>  
    statement 1;  
    statement 2;  
    statement 3;  
  
    ...  
</script>
```

Dove si scrivono gli script?

8

External



Si scrive un file di funzioni JavaScript che viene incluso nel documento HTML nella parte di <head> o di <body> (spesso al fondo prima di </body>)

```
<script src="myfile.js"></script>
```

- Il codice non è “annegato” all'interno della pagina HTML, può essere condiviso tra più script e la manutenzione è più semplice
- Si può sfruttare il caching del browser

Dove si scrivono gli script?

9

Internal

Si inserisce codice JavaScript (istruzioni e funzioni) nella parte di `<head>` o di `<body>` all'interno dei tag `<script> ... </script>`

Inline

Si inserisce codice JavaScript direttamente nei tag HTML (nel caso di programmazione ad eventi, ma oggi questa scelta è **deprecata**)

Quando vengono eseguiti?

10

- **Dipende...**
 - Quando il **browser carica una pagina**
 - se trova **sequenze di istruzioni** le manda in esecuzione
 - se trova **definizioni di funzioni** ne fa il parsing per controllarne la correttezza sintattica
 - Terminato il caricamento della pagina, quando il **browser “sente” un evento**
 - se trova una funzione o un gestore di evento collegati a quell'evento, li esegue

JavaScript: linguaggio interpretato

11

*JavaScript is a lightweight **interpreted programming language**. The web browser receives the JavaScript code in its original text form and runs the script from that.*

*From a technical standpoint, most modern JavaScript interpreters actually use a technique called **just-in-time compiling to improve performance**; the JavaScript source code gets compiled into a faster, binary format while the script is being used, so that it can be run as quickly as possible. However, JavaScript is **still considered an interpreted language, since the compilation is handled at run time, rather than ahead of time**.*

ECMAScript (core language)

12

Definisce la sintassi, i tipi di dato, le istruzioni, le parole riservate, gli operatori, gli oggetti predefiniti, ...

JavaScript: linguaggio dinamico

13

Solo gli **errori sintattici** vengono riconosciuti staticamente prima dell'esecuzione

Gli altri errori vengono riconosciuti a runtime durante l'esecuzione, e non sempre...

I browser mettono a disposizione strumenti per capire cosa sta succedendo...

JavaScript: commenti e ;

14

Le istruzioni sono separate tra loro dal carattere ;
(obbligatorio solo per più istruzioni sulla stessa riga, ma suggerito)

Le parentesi graffe {...} permettono di definire i blocchi di istruzioni

Commenti

es. *// questo è un commento su una sola linea*

es. */* questo è un commento
multilinea */*

es. *<!-- questo è un commento in HTML*

JavaScript: variabili

15

In passato **non era obbligatorio** dichiarare le variabili, ma questa pratica oggi è deprecata: se il codice è in **strict mode** (`"use strict"`) genera un errore

La direttiva `"use strict"` separa il codice *old-style* da quello *modern-style*

A partire da ECMAScript 2015 questa direttiva è sottintesa in molti nuovi costrutti

JavaScript: variabili

16

Per dichiarare una variabile si usano **let**, **const**, **var**

- **let** declares a **block scope local** variable, optionally initializing it to a value
- **const** declares a **read-only** named constant
- **var** declares a **global** or **local** variable, optionally initializing it to a value

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements>

<https://javascript.info/variables>

<https://javascript.info/var>

JavaScript: variabili

17

I nomi delle variabili possono iniziare con `_` `$` o con una **lettera** e non possono iniziare con un numero

Non si possono usare le parole riservate

JavaScript è **case-sensitive**

`let name` **!=** `let Name`

JavaScript: variabili

18

JavaScript è **tipato dinamicamente**, la stessa variabile può essere usata per dati di tipo diverso, senza avere errori

```
message = "Hello world";  
message = 12345;  
message = true;
```

JavaScript: tipi primitivi

19

| Tipo | Descrizione | Esempi |
|-----------|--|---|
| undefined | Valore speciale che rappresenta "value is not assigned" | |
| null | Valore speciale che rappresenta "nothing", "empty" o "value unknown" | |
| boolean | true o false | |
| number | Rappresentazione su 64bit, floating point, limitati da $\pm(2^{53}-1)$ | Infinity (es. 5/0), NaN (es. 0/0), Number.PI |
| bigint | Per rappresentare i numeri di lunghezza arbitraria | bigint = 12345678901234567890123456789012 34567890n |
| string | Sequenza di caratteri delimitata da apici (di 3 tipi) | "Hello world" 'Hello world' `Hello world` |
| symbol | Chiave univoca per una proprietà | var prop1 = Symbol("key") |

<https://javascript.info/types>

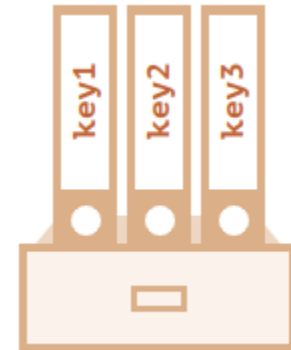
JavaScript: oggetti

20

Gli oggetti hanno **proprietà** definite come coppie **key: value**

```
let person = {  
  firstname: "Alice",  
  lastname  : "Smith",  
  email     : "alice@email.com"  
};
```

```
let empty_object = {};
```



JavaScript: oggetti

21

Si può accedere alle proprietà con

- **dot notation** “.”

```
a = object_name.property_name;
```

- **mediante chiave**

```
a = object_name['property_name'];
```

Se il nome della proprietà non è un identificatore valido, per es. contiene uno spazio oppure altri caratteri speciali, è indispensabile la seconda notazione:

```
myObject.first name           // non valido  
myObject['first name'] = 'John'; // ok
```

JavaScript: oggetti

22

Si possono **aggiungere** e **rimuovere** le **proprietà** di un oggetto dinamicamente

```
person.age = 25; delete person.email;
```

Se si accede a una **proprietà non definita**, il valore di ritorno è `undefined`

Si può accedere a tutte le proprietà di un oggetto con il ciclo **for ... in**

```
for (let key in person) {  
    alert( key + " " + person[key]);  
}
```

<https://javascript.info/object>

JavaScript Object Notation

23

Meglio nota come **JSON**, indica un formato testo usato per lo **scambio dei dati**

La **sintassi** è molto **simile** a quella degli **oggetti**

I nomi delle **proprietà** devono sempre essere scritti tra **doppi apici**

... ne ripareremo...

JavaScript: Array

24

Gli array sono **collezioni ordinate di dati** accessibili mediante indice

In **JavaScript** (come in PHP) gli elementi di un array possono essere di **tipo diverso**

```
let array = [];
```

```
let array = New Array() // poco usato
```

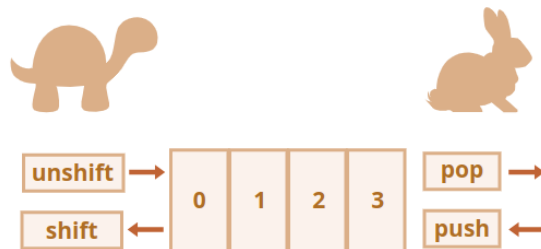
```
let num = ['zero', 'uno', 'due', 'tre', 'quattro'];
```

```
let misc = ['ciao', 10, true, ['M', 'F'], NaN];
```


JavaScript: Array

25

- `array.push(value)` - aggiunge un elemento al fondo
- `array.pop()` - prende l'ultimo elemento
- `array.at(-1)` - prende l'ultimo elemento
- `array.unshift(value)` - aggiunge un elemento in testa
- `array.shift()` - prende il primo elemento
- `array.at(0)` - prende il primo elemento



<https://javascript.info/array>

JavaScript: Array

26

- **array.length** - restituisce la “lunghezza” dell’array

```
let students = [];  
students[40] = "Rossi";  
console.log(students.length); ???
```

- **array.toString()** - restituisce un stringa con gli elementi dell’array separati da una virgola

JavaScript: Map

27

Tipo di dato **built-in** che consente di memorizzare collezioni di coppie non ordinate **<key: value>**

- **new Map()** - crea una nuova mappa vuota
- **map.set(key, value)** - memorizza la coppia **<key: value>**
- **map.get(key)** - restituisce il valore associato a key
- **map.has(key)** - ritorna true se key esiste, false altrimenti
- **map.delete(key)** - rimuove la coppia **<key: value>**
- **map.clear()** - cancella i dati nella mappa
- **map.size()** - ritorna il numero di elementi nella mappa

JavaScript: Set

28

Tipo di dato **built-in** che consente di memorizzare collezioni di valori **senza duplicati**

- **new Set([iterable])** - crea un set e, in presenza di un parametro iterabile, lo inizializza
- **set.add(value)** – aggiunge un valore al set
- **set.delete(value)** - cancella un valore dal set, restituisce true se l'operazione ha successo, false altrimenti
- **set.has(value)** – ritorna true se il valore esiste, false altrimenti
- **set.clear()**– cancella i valori dal set
- **set.size()** – ritorna la dimensione del set

Oggetti built-in: Date

29

Offre proprietà e metodi per lavorare con tempo e date

```
let oggi = new Date();  
  
let date = oggi.getDate();  
let day  = oggi.getDay();  
let full = oggi.getFullYear();  
let hour = oggi.getHours();  
  
let time = oggi.getTime();  
restituisce i millisecondi trascorsi dalla mezzanotte  
del 1/1/1970
```

Oggetti built-in: Math

30

Offre proprietà e metodi per le operazioni matematiche

```
let min  = Math.min(x,y);  
let sqrt = Math.sqrt(x);  
let abs  = Math.abs(x);  
let rnd  = Math.random();  
  
let  $\pi$  = Math.PI;  
let e   = Math.E;
```

Oggetti built-in: String

31

Offre proprietà e metodi per manipolare testo

```
let msg = "Hello world";  
let msg = 'Hello world';  
let msg = `Hello world`;  
  
let len = a.length;  
let chr = a.charAt(n);  
let off = a.indexOf(substr);  
let new = a.replace(espr1, espr2);  
  
// restituisce una sottostringa  
let substr = a.slice(start, end);  
  
// converte una stringa in un array  
let arr = a.split("separator");
```

JavaScript: function

32

```
function fname(param1, param2,...) {  
    // some JavaScript code to be executed  
    return result;  
}  
  
function square(x) { return x ** 2; }  
let a = square(4);
```

Le variabili dichiarate all'interno della funzione sono **locali**

Le variabili che vengono usate (ma non dichiarate) all'interno di una funzione sono **globali**

JavaScript: arrow function

33

```
let fname = (param1, param2, ...) =>  
expression;  
  
let square = (x) => x ** 2;  
let a = square(4);
```

Notazione sintetica per scrivere le funzioni in JavaScript

Viene creata una funzione fname che, presi in input i valori dei suoi parametri, calcola l'espressione a destra della freccia e ritorna il risultato

JavaScript: eventi

34

Esempi di eventi

- L'utente **clicca** con il mouse su un elemento
- L'utente **posiziona il mouse** su un elemento
- L'utente **digita un tasto** sulla tastiera
- L'utente **riempie un campo** di testo
- La **pagina viene caricata** completamente
- L'**immagine viene caricata** completamente
- Il **form HTML viene inviato** ad un server

JavaScript: eventi

35

load
unload

change
reset
submit

blur
focus

click

mousedown
mouseup
mouseover
mouseout

keydown
keypress
keyup

<https://developer.mozilla.org/en-US/docs/Web/Events>

JavaScript: eventi

36

Ad ogni **evento** è associato un **elemento HTML** che inizia con **on** cui segue il **nome dell'evento**

Se nel file HTML esiste del codice JavaScript associato a questo elemento HTML, il codice viene eseguito quando si verifica l'evento

Esempio (oggi deprecato, vedremo il modern-style)

```

```

JavaScript: input/output

37

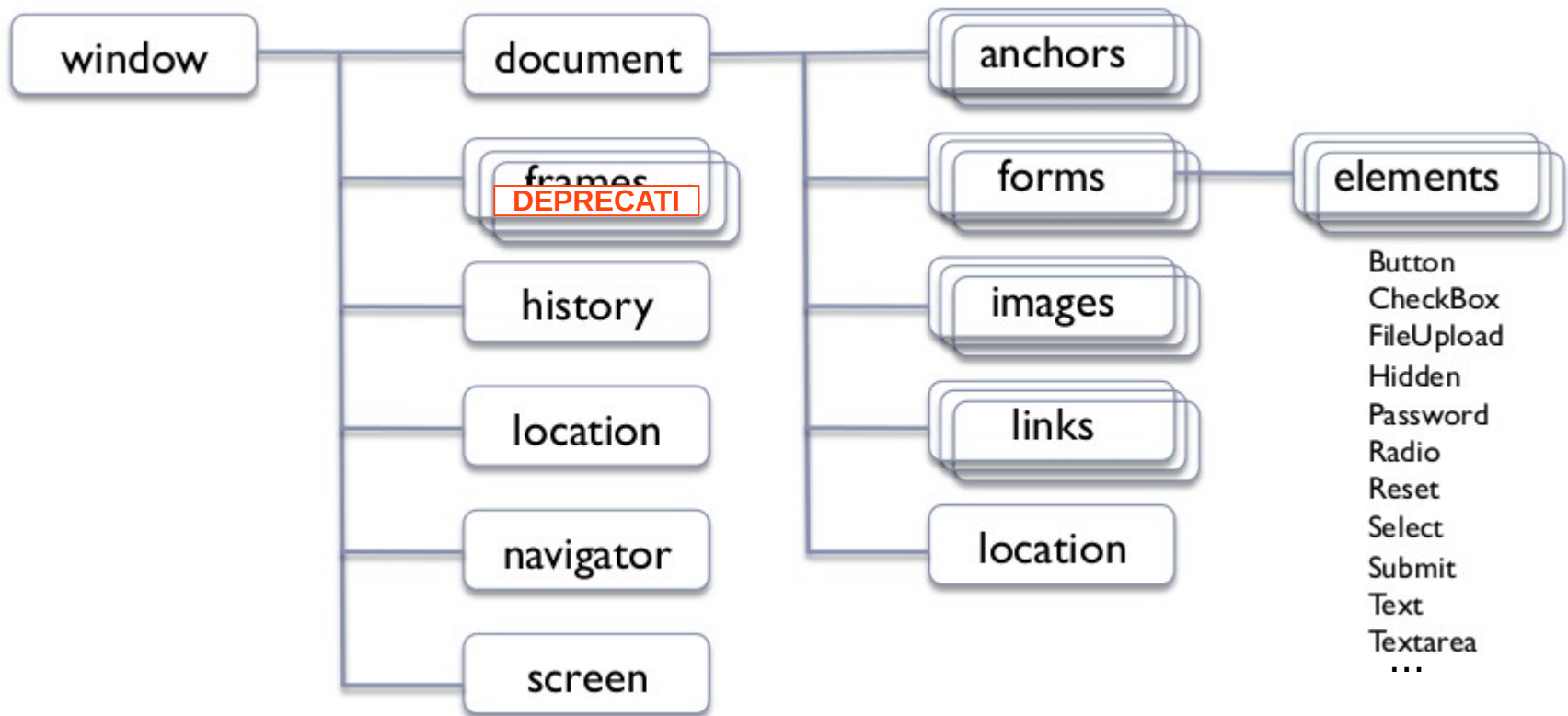
JavaScript **non ha propri costrutti di input e output**

Se JavaScript è integrato in un browser, si collega alle applicazioni tramite interfacce chiamate **BOM** (Browser Object Model) e **DOM** (Document Object Model)

Anticipiamo l'oggetto **window** (BOM) che permette di vedere alcuni aspetti dell'input/output

JavaScript BOM+DOM

38



Oggetto Window

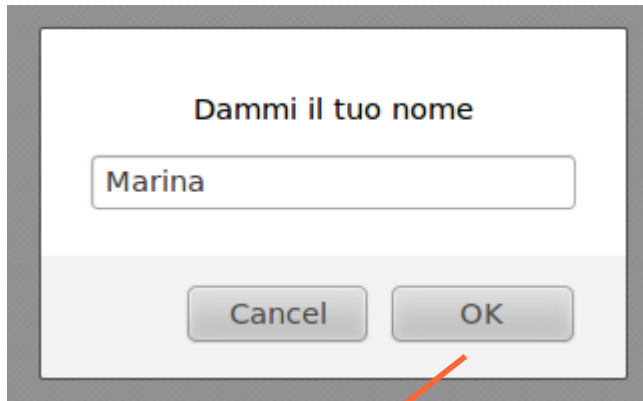
39

- **Rappresenta il browser**
- Permette di
 - realizzare operazioni di input/output mediante **widget modali** che bloccano l'esecuzione dello script fino a quando non vengono chiusi
 - aprire nuove finestre (popup)
 - chiudere le finestre aperte dallo script
 - ...

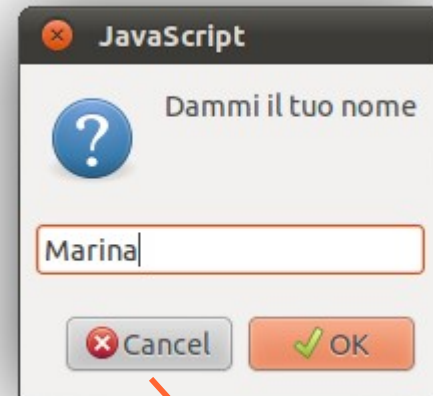
JavaScript: prompt

40

```
window.prompt("Dammi il tuo nome", "")
```



valore in input

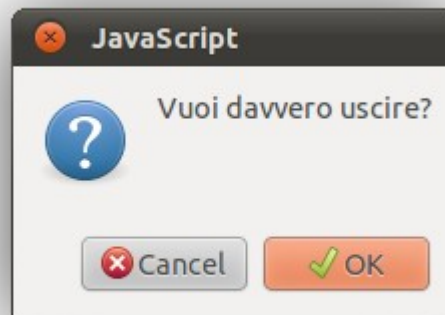


"" (null)

JavaScript: confirm

41

```
window.confirm("Vuoi davvero uscire?")
```



JavaScript: alert

42

`window.alert("Hai restituito ...")`



Altri oggetti del browser

43

- Navigator

descrive il **browser** che sta facendo la richiesta HTTP

- Location

descrive la **barra degli indirizzi** e può essere usato per operazioni di redirect verso nuove pagine

- Screen

descrive lo **schermo** dell'utente

Uso della console JS

44

