

Parte 1

Il linguaggio JavaScript

JS

Ilya Kantor

Costruito a 6 luglio 2025

L'ultima versione di questo tutorial è reperibile a <https://it.javascript.info>.

Lavoriamo costantemente per migliorare il tutorial. Se trovi qualsiasi errore, segnalalo sul [nostro github](#).

- [Introduzione](#)
 - [Introduzione a JavaScript](#)
 - [Manuali e Specifiche](#)
 - [Code editor](#)
 - [Developer console](#)

Impareremo JavaScript, iniziando dalle basi e passando a concetti avanzati come OOP.

Ci concentreremo principalmente sul linguaggio, con un minimo di annotazioni riguardo gli ambienti di sviluppo.

Introduzione

Studieremo il linguaggio JavaScript e l'ambiente di sviluppo.

Introduzione a JavaScript

Vediamo cosa rende JavaScript così speciale, cosa è possibile ottenere tramite il suo utilizzo e tutte le tecnologie che possono essere applicate per renderlo adatto ad ogni necessità.

Cos'è JavaScript?

JavaScript è stato creato con lo scopo di “dare vita alle pagine web”.

I programmi che sfruttano questo linguaggio vengono chiamati *script*. Possono essere scritti direttamente nel documento HTML ed eseguiti in automatico al caricamento della pagina.

Gli script vengono scritti ed eseguiti come testo semplice. Per questo non richiedono alcuna fase di preparazione o compilazione per essere eseguiti.

Sotto questo aspetto, JavaScript è molto differente da un altro linguaggio chiamato [Java](#) ↗ .

Perché si chiama JavaScript?

In origine JavaScript aveva un altro nome: “LiveScript”. In quel periodo Java era molto popolare, per questo si è pensato che identificare Javascript come il “fratello minore” di Java potesse aiutare alla sua diffusione.

Evolvendosi, JavaScript è diventato un linguaggio completamente indipendente, le cui specifiche sono definite da [ECMAScript](#) ↗ , e adesso non ha quasi nulla in comune con Java.

Attualmente, JavaScript può essere eseguito non solo nei browser, ma anche nei server web e in altri ambienti che supportano il [motore JavaScript](#) ↗ (JavaScript engine).

Il browser ha un suo motore JavaScript integrato, chiamato alle volte “JavaScript Virtual Machine”.

Esistono altri motori JavaScript, tra cui:

- [V8](#) – per Chrome e Opera.
- [SpiderMonkey](#) – per Firefox.
- ...Ci sono altri codenames come “Chakra” per IE, “JavaScriptCore”, “Nitro” e “SquirrelFish” per Safari, etc.

I nomi citati sopra possono essere utili da ricordare, poiché si possono trovare spesso in articoli che trattano di sviluppo web. Anche noi li useremo. Ad esempio, se “una caratteristica X è supportata da V8”, probabilmente funzioneranno senza problemi in Chrome e Opera.

i Come funzionano questi motori?

Il funzionamento di questi motori è complicato, ma i concetti alla base sono semplici.

1. I motori (integrati nei browser) leggono (“analizzano”) lo script.
2. Successivamente convertono (“compilano”) lo script nel linguaggio della macchina.
3. Infine il “codice macchina” viene eseguito, molto rapidamente.

Il motore ottimizza il codice ad ogni passaggio del processo, anche durante l'esecuzione dello script già compilato, quando ne analizza il flusso dati. Nonostante tutto l'esecuzione dello script risulta essere molto veloce.

Cosa può fare JavaScript a livello browser?

JavaScript, al giorno d'oggi, è un linguaggio di programmazione “sicuro”. Non consente alcun accesso di basso livello alla memoria o alla CPU. Questo perché è stato creato con lo scopo di funzionare nei browser, che non richiedono questi tipi di privilegi.

Le capacità di JavaScript dipendono molto dall'ambiente in cui lo si esegue. Ad esempio, [Node.js](#) supporta funzioni che consentono a JavaScript di scrivere/leggere file, eseguire richieste web, etc.

Integrato nel browser Javascript può fare qualsiasi cosa legata alla manipolazione della pagina, all'interazione con l'utente e con il server.

Ad esempio, è possibile:

- Aggiungere HTML alla pagina, cambiare il contenuto esistente, modificare lo stile.
- Reagire alle azioni dell'utente, click del mouse, movimenti del cursore, input da tastiera.
- Inviare richieste al server tramite la rete, caricare e scaricare file (con l'ausilio di [AJAX](#) e [COMET](#)).
- Prelevare e impostare cookies, interrogare l'utente, mostrare messaggi.
- Memorizzare i dati client-side ("memorizzazione locale").

Cosa NON può fare JavaScript a livello browser?

Per la sicurezza dell'utente, le possibilità di JavaScript nel browser sono limitate. L'intento è di prevenire che una pagina "maligna" tenti di accedere alle informazioni personali o di danneggiare i dati degli utenti.

Esempi di queste restrizioni possono essere:

- JavaScript, in una pagina web, non può leggere o scrivere in qualsiasi file nell'hard disk, né copiare o eseguire programmi. Non ha accesso diretto alle funzioni del sistema operativo.

I moderni browser gli consentono di lavorare con i file, sempre con un accesso limitato e comunque solo se il comando proviene da utente, come il "dropping" di un file nella finestra del browser, o con la selezione tramite il tag `<input>`.

Ci sono anche funzionalità che consentono di interagire con la camera/microfono e altri dispositivi, ma in ogni caso richiedono il permesso esplicito dell'utente. Quindi una pagina con JavaScript abilitato non può attivare la web-cam di nascosto, osservare i nostri comportamenti e inviare informazioni alla [CIA](#).

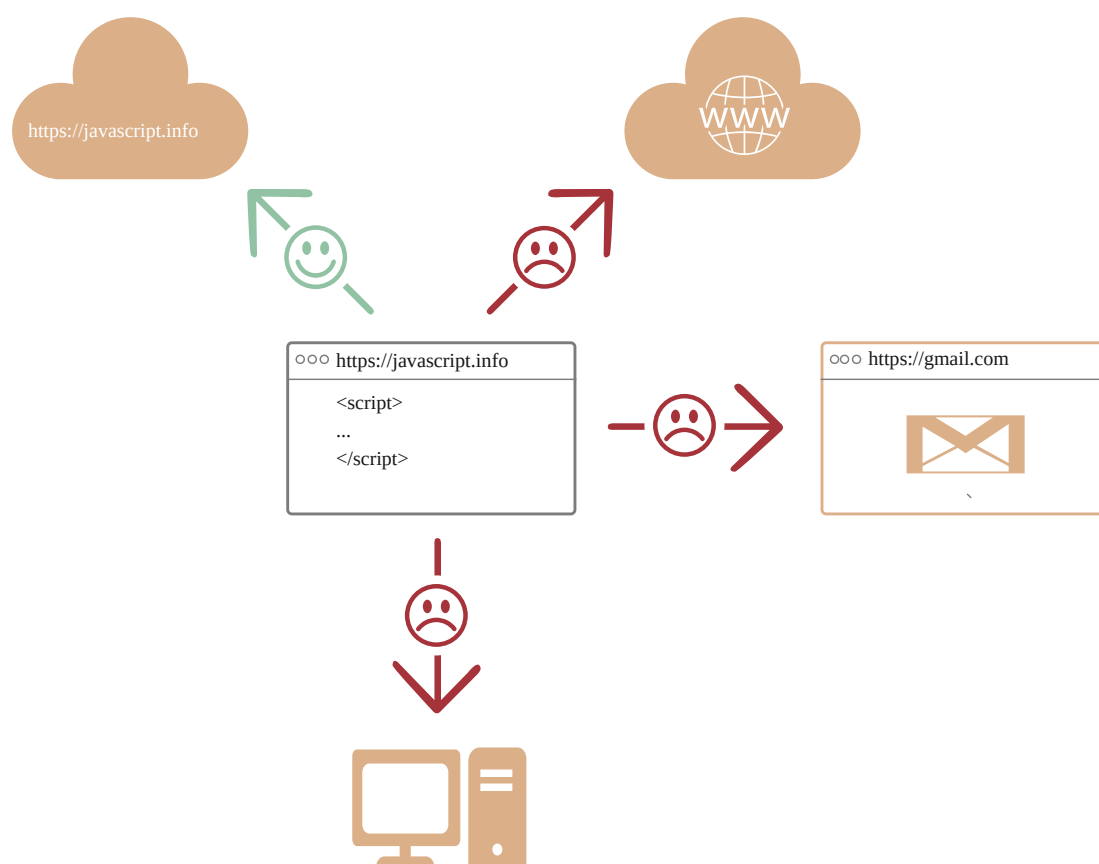
- Pagine o schede diverse generalmente non sono a conoscenza dell'esistenza delle altre. In certi casi, tuttavia, può capitare; ad esempio quando una finestra ne apre un'altra tramite JavaScript. Ma anche in questo caso, il codice JavaScript non può accedere all'altra pagina se non appartiene allo stesso sito (stesso dominio, protocollo o porta).

Questa viene definita la "Same Origin Policy" ("Politica di Appartenenza alla Stessa Origine"). Per poter aggirare questo limite, *entrambe le pagine* devono contenere uno speciale codice JavaScript che consente di gestire lo scambio di dati.

Questa limitazione è sempre dovuta alla sicurezza dell'utente. Una pagina proveniente da `http://anysite.com` che è stata aperta da un utente, ad

esempio, non deve essere in grado di accedere ad un'altra scheda del browser con l'URL `http://gmail.com` e rubarne le informazioni.

- JavaScript può facilmente comunicare con il server da cui la pagina proviene. Ma la sua abilità di ricevere dati da altri siti/domini è limitata. Sebbene sia possibile, sono richieste esplicite autorizzazioni (passate tramite HTTP headers) dall'indirizzo remoto. Ancora una volta, una limitazione dovuta alla sicurezza.



Queste limitazioni non si pongono se JavaScript viene eseguito fuori dal browser, ad esempio in un server. I browser moderni permettono l'installazione di plugin ed estensioni che consentono di estendere vari permessi.

Cosa rende JavaScript unico?

Ci sono almeno *tre* cose che rendono JavaScript così unico:

- Completa integrazione con HTML/CSS.
- Operazioni semplici vengono eseguite semplicemente.
- Supportato dai maggiori browser ed integrato di default.

JavaScript è l'unica tecnologia in ambiente browser che combina queste tre caratteristiche.

Questo rende JavaScript unico. Ed è il motivo per cui è lo strumento più diffuso per creare interfacce web.

Quando si ha in programma di imparare una nuova tecnologia, è fondamentale verificare le sue prospettive. Quindi diamo uno sguardo alle nuove tendenze che includono nuovi linguaggi e tecnologie.

Linguaggi “oltre” JavaScript

La sintassi di JavaScript non soddisfa le necessità di tutti. Alcune persone necessitano di caratteristiche differenti.

Questo è prevedibile, poiché i progetti e i requisiti sono diversi da persona a persona.

Recentemente, per questo motivo, sono nati molti nuovi linguaggi che vengono *convertiti* in JavaScript prima di essere eseguiti nel browser.

Gli strumenti moderni rendono la conversione molto veloce e pulita, consentendo agli sviluppatori di programmare in un altro linguaggio e di auto-convertirlo *under the hood*.

Esempi di alcuni linguaggi:

- [CoffeeScript](#) ➦ è un linguaggio che introduce una sintassi semplificata che consente di scrivere codice più leggibile. Amato dagli sviluppatori provenienti da Ruby.
- [TypeScript](#) ➦ si occupa di aggiungere la “tipizzazione”, per semplificare lo sviluppo e supportare sistemi più complessi. E' stato sviluppato da Microsoft.
- [Flow](#) ➦ anche'esso aggiunge la tipizzazione dei dati, ma in un modo differente. Sviluppato da Facebook.
- [Dart](#) ➦ è un linguaggio autonomo che possiede il suo motore, che esegue in ambienti esterni al browser (come mobile apps). E' stato introdotto da Google come alternativa a JavaScript, ma attualmente i browser richiedono la conversione in JavaScript, proprio come i precedenti.
- [Brython](#) ➦ è un *transpiler*, scritto in Python, che consente di scrivere applicazioni in quest'ultimo senza utilizzare JavaScript.
- [Kotlin](#) ➦ è un moderno, conciso e sicuro linguaggio di programmazione mirato ai browsers o a Node.

Ce ne sono molti altri. Ovviamente, per comprendere cosa stiamo facendo, se utilizziamo uno di questi linguaggi dovremmo altresì conoscere JavaScript.


Riepilogo

- JavaScript è stato creato specificamente per i browser, ma attualmente viene utilizzato con efficacia in molti altri ambienti.
- Attualmente, per quanto riguarda lo sviluppo del web, JavaScript si trova in una posizione unica grazie ad una completa integrazione con HTML/CSS.
- Ci sono molti linguaggi che possono essere “convertiti” in JavaScript; essi forniscono le stesse funzionalità e risolvono gli stessi problemi. E' fortemente consigliato di leggere brevemente le funzionalità di alcuni di essi, dopo aver studiato e compreso JavaScript.


Manuali e Specifiche


Questo libro è un *tutorial*. L'obiettivo è quello di aiutarti ad apprendere il linguaggio gradualmente. Una volta che avrai familiarizzato con le basi avrai bisogno di ulteriori risorse.

Specifiche

La [specifica ECMA-262](#)  contiene informazioni più dettagliate, approfondite e formalizzate riguardanti JavaScript. E' la definizione stessa del linguaggio.

Iniziare a studiare dalla specifica può risultare difficile. Se avete bisogno di una fonte affidabile e formale riguardante i dettagli del linguaggio, la specifica è il posto in cui cercare. Ma non è una risorsa comoda da consultare per i problemi di tutti i giorni.

Ogni anno viene rilasciata una nuova specifica. Di queste pubblicazioni, è possibile trovare l'ultima bozza a <https://tc39.es/ecma262/> .

Per leggere delle più recenti caratteristiche, incluse quelle considerate “quasi standard” (definite “stage 3”), potete consultare <https://github.com/tc39/proposals> .

Inoltre, se state sviluppando in ambiente browser, ci sono ulteriori specifiche che verranno analizzate nella [seconda parte](#) del tutorial.

Manuali

- **MDN (Mozilla) JavaScript Reference** è il manuale principale, corredato di spiegazioni teoriche, esempi ed altre informazioni utili. E' ottimo per avere informazioni dettagliate riguardo le funzioni e altre caratteristiche del linguaggio.

Può essere consultato a <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference> .

Tuttavia, spesso è meglio fare una ricerca su internet. E' sufficiente cercare "MDN", seguito dal termine da ricercare, e.g. <https://google.com/search?q=MDN+parseInt> per ricercare la funzione `parseInt`, oppure frasi come "RegExp MSDN" o "RegExp MSDN javascript".

Può essere consultato al link <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference> .

- **MSDN** – Manuale Microsoft con molte informazioni, tra cui su JavaScript (a cui viene fatto riferimento con il termine JScript). Se si ha bisogno di ottenere qualche informazione specifica per Internet Explorer, meglio consultare la guida: <http://msdn.microsoft.com/> .

Tabelle di compatibilità

JavaScript è un linguaggio che muta costantemente, con nuove funzionalità che vengono aggiunte regolarmente.

Per verificare il loro supporto da parte dei browser, si possono consultare:

- <http://caniuse.com> – per le tabelle di supporto di ogni caratteristica, ad esempio per visualizzare le funzioni di crittografia: <http://caniuse.com/#feat=cryptography> .
- <https://kangax.github.io/compat-table> – una tabella con le caratteristiche del linguaggio e i motori che le supportano.

Tutte le risorse elencate finora sono utili nello sviluppo di tutti i giorni, in quanto contengono ottime informazioni riguardo ai dettagli del linguaggio, il loro supporto ecc.

Ti consiglio quindi di ricordartele (in alternativa puoi consultare questa pagina), nel caso dovessi avere bisogno di informazioni dettagliate riguardo a qualche caratteristica particolare.

Code editor

Un code editor è il posto in cui i programmatori passano la maggior parte del loro tempo.

Ci sono due principali tipi di code editor: IDE ed editor semplici. Molte persone si trovano bene a sceglierne uno per entrambe le categorie.

IDE

Il termine [IDE](#) (Integrated Development Environment) descrive un potente editor che copre lo sviluppo dell'intero progetto. Come anche il nome suggerisce, non è un semplice editor, ma un "ambiente di sviluppo" scalabile, con molte funzionalità.

Un IDE carica il progetto (possono essere molti file), consente la navigazione tra i file, fornisce il completamento automatico basandosi sull'intero progetto (non sul singolo file), può essere integrato con sistemi di gestione di versione (come [git](#)), un ambiente dedicato al test e altre funzionalità a livello del progetto.

Se non hai ancora considerato di scegliere un IDE, dai un'occhiata a queste alternative:

- [Visual Studio Code](#) (*cross-platform*, gratuito).
- [WebStorm](#) (*cross-platform*, a pagamento).

Per Windows, c'è anche l'editor "Visual Studio", da non confondere con "Visual Studio Code". "Visual Studio" è un potente editor (a pagamento) disponibile solo per Windows, ottimo per le piattaforme .NET. E' disponibile anche una versione gratuita: ([Visual Studio Community](#)).

Molti IDE sono a pagamento, ma offrono un periodo di prova gratuito. Solitamente il loro costo è trascurabile se paragonato allo stipendio di uno sviluppatore qualificato; è quindi importante scegliere il migliore in base alle proprie esigenze.

Editor semplici

Gli "editor Semplici" non sono potenti come gli IDE ma sono molto veloci, eleganti e semplici.







Sono principalmente utilizzati per aprire un file e modificarlo rapidamente.

La principale differenza tra gli editor semplici e un IDE è che quest'ultimo lavora a vari livelli del progetto, carica molti più dati quando viene aperto, analizza la

struttura del progetto e così via. Un editor semplice è molto più veloce poiché necessita solo del file.

In pratica, tuttavia, gli editor semplici possono avere molti plugin, tra cui la sintassi a livello directory e l'autocompletamento, quindi non ci sono delle differenze ben definite tra un editor semplice e un IDE.

Meritano attenzione le seguenti opzioni:

- [Atom](#)  (*cross-platform*, gratuito).
- [Visual Studio Code](#)  (*cross-platform*, gratuito).
- [Sublime Text](#)  (*cross-platform*, con prova gratuita).
- [Notepad++](#)  (Windows, gratuito).
- [Vim](#)  e [Emacs](#)  sono particolarmente carini se si sanno utilizzare.


Non intestarditevi

Gli editor elencanti sopra sono quelli che io e i miei amici, che considero buoni sviluppatori, abbiamo utilizzato senza problemi per molto tempo.

Ci sono altri grandi editor nel nostro grande mondo. Scegli quello che più ti si addice.

La scelta di un editor, come pure di altri strumenti, è individuale e dipende dai progetti, dalle abitudini e preferenze personali.

Developer console

Il codice è incline a contenere errori. E' molto probabile che tu commetta errori... Di cosa sto parlando? *Sicuramente* commetterai errori, sempre che tu sia umano, e non un [robot](#) .

In un browser però, di default l'utente non può vedere gli errori. Quindi, se qualcosa non funziona nello script, non saremo in grado di capire quale sia il problema e sistemarlo.

Per poter visualizzare gli errori e ricevere altre informazioni utili riguardo gli script, i browser integrano degli "strumenti di sviluppo", in inglese "developer tools" o più semplicemente "DevTools".

Molti sviluppatori preferiscono utilizzare Chrome o Firefox poiché questi browser incorporano i migliori strumenti per lo sviluppo. Anche gli altri browser hanno gli strumenti per lo sviluppo, talvolta con caratteristiche speciali, ma più che altro inseguono le caratteristiche di Chrome e Firefox. In genere gli sviluppatori

hanno un browser “preferito” e utilizzano gli altri solo quando un problema è specifico di quel browser.

Gli strumenti per lo sviluppo sono molto potenti e possiedono molte funzionalità. Per iniziare, dobbiamo capire come accedervi, come individuare gli errori e come eseguire comandi JavaScript.

Google Chrome

Apri la pagina [bug.html](#).

C'è un errore nel codice JavaScript. E' nascosto agli occhi di un normale utente, quindi dobbiamo aprire gli strumenti di sviluppo per trovarlo.

Premi `F12`, oppure, se sei su Mac, utilizza `Cmd+Opt+J`.

Gli strumenti di sviluppo, di default, si apriranno nella scheda Console.

Assomiglierà a qualcosa di simile a questo:

Immagine `"/article/devtools/chrome.png"` corrotta

Il look esatto degli strumenti di sviluppo dipenderà dalla tua versione di Chrome.

Nel tempo potrebbe cambiare un po', ma dovrebbe essere comunque molto simile.

- Qui possiamo notare il messaggio d'errore in rosso. In questo caso, lo script contiene il comando “lalala” non riconosciuto.
- Sulla destra, c'è il link cliccabile della sorgente `bug.html:12` con il numero della linea in cui si è verificato l'errore.

Sotto il messaggio d'errore, c'è il simbolo blu `>`. Questo indica la “riga di comando” in cui possiamo digitare dei comandi JavaScript. Premendo `Enter` il comando viene eseguito (`Shift+Enter` per inserire comandi multi-linea).

Ora possiamo vedere gli errori, e questo è sufficiente per iniziare. Ritorneremo sugli strumenti di sviluppo più avanti e analizzeremo il debugging più in profondità nel capitolo [Debugging in the browser](#).

Input multi-riga

Di solito, quando inseriamo una riga di codice nella console e premiamo il tasto `Enter`, questa viene eseguita.

Per inserire più righe premi `Shift+Enter`, in questo modo puoi inserire lunghe porzioni di codice Javascript.

Firefox, Edge, and others

Molti altri browser utilizzano `F12` per aprire gli strumenti di sviluppo.

Anche l'aspetto è molto simile. Quando avrai imparato come utilizzare uno di questi strumenti (puoi iniziare con quelli di Chrome), potrai facilmente utilizzare anche gli altri.

Safari

Safari (Mac browser, non supportato da Windows/Linux) è un pò speciale in questo ambito. E' necessario attivare prima il "Menu di Sviluppo".

Apri le Impostazioni e vai sul pannello "Avanzate". In basso troverai un'opzione da spuntare:

Immagine `"/article/devtools/safari.png"` corrotta

Adesso tramite `Cmd+Opt+C` puoi attivare e disattivare la console. Inoltre noterai che un nuovo menu "Sviluppo" è apparso. Esso contiene molti comandi e opzioni.

Riepilogo

- Gli strumenti di sviluppo (Developer Tools o DevTools) ci consentono di individuare gli errori, eseguire comandi, esaminare variabili e molto altro.
- Possono essere attivati con `F12` nella maggior parte dei browser in Windows e Linux. Chrome su Mac `Cmd+Opt+J`, Safari: `Cmd+Opt+C` (avendolo precedentemente abilitato).

Ora in nostro ambiente di sviluppo è pronto. Nella prossima sezione inizieremo ad analizzare JavaScript.