

**Formazione
specialistica
per sviluppatore
front-end**





Hello!

Alessandro Pasqualini

Full-stack developer, appassionato di
programmazione e sviluppo software in generale, amo
le serie tv e i gatti

1

Javascript



Cos'è Javascript?

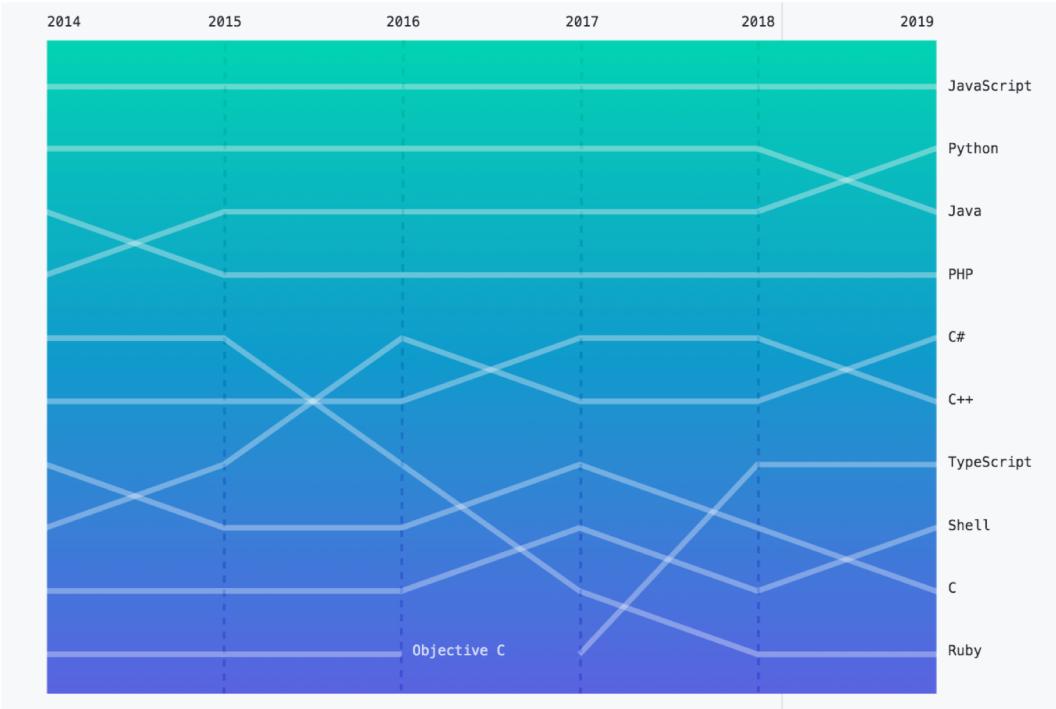
Javascript è un linguaggio di programmazione che permette di aggiungere interattività e comportamenti personalizzati alla pagina web.

Le tecnologie del web:

- HTML definisce la struttura della pagina
- CSS definisce lo stile della pagina
- JS definisce il comportamento personalizzato della pagina



Cos'è Javascript?



<https://octoverse.github.com/>



Cos'è Javascript?

01	microsoft/vscode	19.1k
02	MicrosoftDocs/azure-docs	14k
03	flutter/flutter	13k
04	firstcontributions/first-contributions	11.6k
05	tensorflow/tensorflow	9.9k
06	facebook/react-native	9.1k
07	kubernetes/kubernetes	6.9k
08	DefinitelyTyped/DefinitelyTyped	6.9k
09	ansible/ansible	6.8k
10	home-assistant/home-assistant	6.3k

<https://octoverse.github.com/>



Javascript lato server

Javascript è uno dei linguaggi più usati al mondo.

"Di recente" è stato "trasformato" anche in un linguaggio lato server: **nodejs**.

Nodejs permette di usare le stesse tecnologie per il web per realizzare la cosiddetta business logic dell'applicazione.

“

*Java is to Javascript what Car is
to Carpet*

- Chris Heilmann



Javascript vs Java

Java è un linguaggio completamente diverso da Javascript.

L'unica cosa che hanno in comune è una parte del nome.

Javascript ha avuto negli anni una brutta reputazione: era il responsabile di redirect, popup e il fulcro di tantissimi problemi di sicurezza.

I browser moderni controllano e impediscono a JS tutta una serie di "attività" per la sicurezza dell'utente.



Cosa si può fare con JS?

Accedere al contenuto: selezionare un elemento, accedere alle sue proprietà, accedere al CSS, accedere al contenuto, etc

Modificare un elemento: modificare il contenuto di un elemento, creare elementi "on the fly", cambiare le sue proprietà, cambiare i suoi attributi, cambiare il CSS, etc

Reagire ad eventi: creare del codice che deve essere eseguito se succede qualcosa (evento). Es. il click del mouse, l'hover, drag & drop etc



Per cosa è usato JS?

Validazione di form: validare il contenuto inserito dall'utente prima di inviare il contenuto al server. (c'è sempre bisogno della validazione anche nel server)

Slideshow/animazioni: mostrare contenuto diverso nello stesso spazio, animare **semplicemente** il contenuto. (Le animazioni posso essere fatte anche con CSS3 ma è più complicato)

Etc...



Per cosa è usato JS?

Ricaricare parte (o tutta) la pagina in modo "asincrono": caricare il contenuto senza che l'utente debba attendere il "refresh della pagina" (Single Page Application)

Filtrare i dati: filtrare i dati nella pagina in base a delle "indicazioni" dell'utente

Testare il browser: ogni browser è diverso e con JS possiamo capire quale browser l'utente sta usando e agire di conseguenza per correggere eventuali problemi



Inserire JS nella pagine

```
<script>  
    // Il mio JS  
</script>  
  
<script src="..."></script>
```

Javascript può essere incluso in due modi:

- Embedded nella pagina (similarmente al tag style)
- Inserendo il link del file JS (similarmente al tag link)



Dove includere il JS nella pagina?

```
<!DOCTYPE html>
<html>
  <head>
    <title>Il mio primo JS</title>
    <script>
      // JS
    <script>
  </head>
  <body>
    <script>
      // JS
    <script>
  </body>
</html>
```

Javascript può essere incluso sia nel tag **head**, oppure nel tag **body**.

Non fa (quasi) differenza.



Non fa davvero differenza?

In realtà una pagina web **fatta bene** deve includere il JS come ultima cosa del tag **body**

Il browser interpreta la pagina dall'inizio alla fine. Se il JS richiede operazione pesanti la pagina verrà non verrà renderizzata finchè non è stato interpretato (e scaricato) tutto il JS.

E' più importante che la pagina si veda bene piuttosto che sia "dinamica". I tempi di interpretazione "massimi" del JS dovrebbero comunque attestarsi intorno 1s. Massimi non medi!



Non fa davvero differenza?

Se il JS è scritto in file esterno (e magari pesa parecchio) se è incluso nel tag head il browser impegnerà risorse per scaricare il JS, privandole ad altre risorse più importanti nell'immediato (es. immagini/CSS)

Se il JS è embedded nella pagina e inserito nel tag head il browser inizierà la sua interpretazione subito appena incontra il tag script, privando risorse al rendering della pagina.



Il JS va sempre inserito nel body

Se il js è esterno oppure embedded ed è posizionato come ultima cosa del body quando il browser lo incontra avrà terminato il rendering della pagina oppure sarà in procinto di farlo e quindi non vengono "sprecate" risorse.

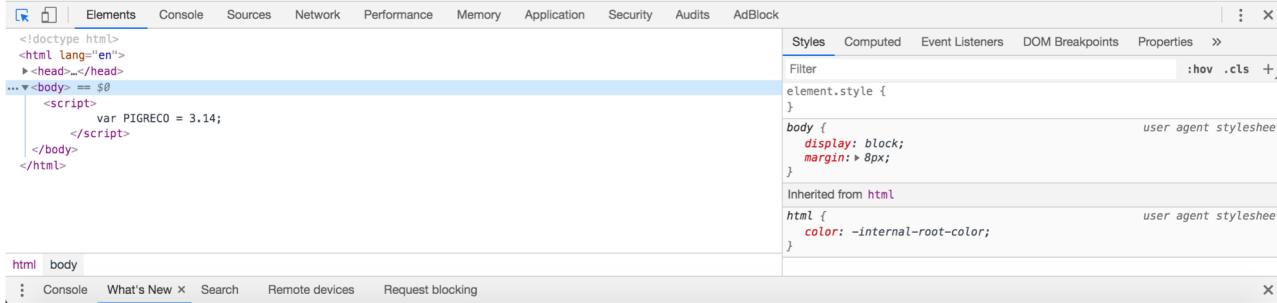
Questa regola generale non si applica ad alcuni script particolari (ad esempio il tracciamento dell'utente o il blocco dei cookies) che richiedono di essere inizializzati prima del contenuto (e del resto del codice JS)

2

Strumenti per sviluppatori



La console di Google Chrome



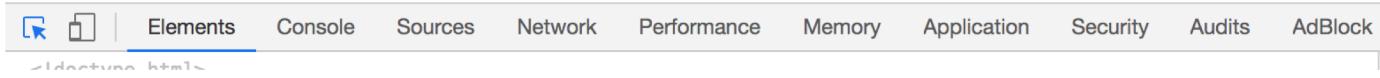
```
<!doctype html>
<html lang="en">
  <head></head>
  <body> == $0
    <script>
      var PIGRECO = 3.14;
    </script>
  </body>
</html>
```

The screenshot shows the Google Chrome DevTools developer console. The top navigation bar includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, Audits, and AdBlock. The Elements tab is active, displaying the DOM structure of a simple HTML page with a single script tag in the body. The right-hand panel shows the Styles tab of the DevTools, which displays the CSS rules applied to the body element. It includes the user agent stylesheet rule for body (display: block; margin: 8px;) and the user agent stylesheet rule for html (color: -internal-root-color;). The bottom of the DevTools interface shows tabs for html and body, and a footer with links for Console, What's New, Search, Remote devices, and Request blocking.

La console di sviluppo (strumenti per sviluppatori) è di fondamentale utilizzo quando si sviluppa (e testa) il frontend di un sito internet.



La console di Google Chrome



Ci sono diversi tab e strumenti messi a disposizione dello sviluppatore.

I principali (quelli che servono a noi) sono:

- Elements
- Console
- Network



Tab Elements

The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The left pane displays the DOM tree:

```
<!doctype html>
<html lang="en">
  <head>...</head>
  ...<body> == $0
    <script>
      var PIGRECO = 3.14;
    </script>
  </body>
</html>
```

The right pane shows the 'Styles' tab with the following CSS rules:

```
element.style { }
body {
  display: block;
  margin: 8px;
}
Inherited from html
html {
  color: -internal-root-color;
}
```

Below the styles, there is an element inspector showing a box model diagram for a body element with a width of 872 and a height of 0. The diagram illustrates the padding (green), border (thin black line), and margin (orange) around the content area (blue).

At the bottom, the DevTools navigation bar includes tabs for 'Console', 'What's New', 'Search', 'Remote devices', and 'Request blocking'.

Qui possiamo ispezionare il codice HTML della pagina e i CSS della pagina.



Sottotab Styles

The screenshot shows the 'Styles' tab in the Chrome DevTools. It displays the following CSS rules:

```
element.style {  
}  
  
body {  
  display: block;  
  margin: ▶ 8px;  
}  
  
Inherited from html  
html {  
  color: -internal-root-color;  
}
```

Below the rules, there is a visual representation of the box model for an element. The box model consists of four parts: margin (orange), border (yellow), padding (green), and content (blue). The total width of the box is 872 pixels, and the height is 0 pixels.

Nella "sottotab" **Styles** possiamo ispezionare le varie regole CSS ed eventualmente applicarne di nuove a **runtime** per verificare il loro comportamento prima di inserirle nel nostro file di stile.



Sottotab Styles

The screenshot shows the 'Styles' tab in the Chrome DevTools. The left pane displays a list of CSS rules for the color 'blue'. The first rule is 'element.style { background: blue; }'. Below it is a list of 'body' rules from the 'user agent stylesheet': 'background-color: blue;', 'color: blue;', 'font-family: sans-serif;', 'font-size: 1em;', 'font-style: normal;', 'font-variant: normal;', 'font-weight: 400;', 'text-decoration: none;', 'text-indent: 0px;', 'text-align: left;', 'text-decoration-color: inherit;', 'text-decoration-line: none;', 'text-decoration-style: solid;', 'text-decoration-width: 0px;', 'text-orientation: upright;', 'writing-mode: normal;', and 'margin: 0px;'. There is also an 'Inherited from html' section with similar rules. The right pane shows the 'Properties' tab with a preview of a blue square.

Nella "sottotab" **Styles** possiamo ispezionare le varie regole CSS ed eventualmente applicarne di nuove a **runtime** per verificare il loro comportamento prima di inserirle nel nostro file di stile.



Sottotab Computed

The screenshot shows the Chrome DevTools interface with the "Computed" tab selected. At the top, there are tabs for "Styles", "Computed", "Event Listeners", "DOM Breakpoints", and "Properties". Below the tabs, there's a visual representation of an element's bounding box with various padding and margin values labeled. A "Filter" input field and a "Show all" checkbox are present. The main list displays the following computed properties:

Property	Value
color	rgb(0, 0, 0)
display	block
height	0px
margin-bottom	8px
margin-left	8px
margin-right	8px
margin-top	8px
width	872px

Nella "sottotab" **Computed** possiamo ispezionare le varie proprietà CSS possedute da uno specifico elemento.

Qui possiamo vedere in base alla gerarchia delle regole CSS cosa il browser ha deciso di applicare all'elemento.



Sottotab Computed

The screenshot shows the Chrome DevTools interface with the "Computed" tab selected. At the top, there's a visual representation of an element's bounding box with various padding and margin values labeled. Below this, a list of CSS properties is shown, grouped by source: element.style, padding-box, user agent stylesheet, and html. The properties listed include background-color (blue), background-image (none), background-origin (padding-box), background-position-x (0%), background-position-y (0%), background-repeat-x, background-repeat-y, background-size (auto), color (rgb(0, 0, 0)), and font-size (16px). The "background-color" property is highlighted with a blue square icon.

Property	Value	Source
background-color	blue	element.style
background-color	rgb(0, 0, 255)	padding-box
color	rgb(0, 0, 0)	user agent stylesheet
font-size	16px	html

Nella "sottotab" **Computed** possiamo ispezionare le varie proprietà CSS possedute da uno specifico elemento.

Qui possiamo vedere in base alla gerarchia delle regole CSS cosa il browser ha deciso di applicare all'elemento.



Tab Console

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The main area displays a message: 'No messages' and 'Expression not available'. On the left, a sidebar lists filtering options: 'No user me...', 'No errors', 'No warnings', 'No info', and 'No verbose'. At the bottom, there's a navigation bar with tabs for 'Console', 'What's New', 'Search', 'Remote devices', and 'Request blocking'.

La console è lo strumento che ci mette a disposizione il browser per avere informazioni, errori, avvisi sul JS della pagina.



Tab Console

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The left sidebar lists log levels: 1 message, 1 error, 0 warnings, 0 info, and 0 verbose. The main console area displays two messages: 'Expression not available' (greyed out) and a red-highlighted 'Uncaught SyntaxError: Invalid or unexpected token' at index.html:12. Below the console, tabs for 'Console', 'What's New', 'Search', 'Remote devices', and 'Request blocking' are visible.

Se ci sono errori di interpretazione il browser ci fornisce il tipo di errore, il file e la linea di dove si è verificato.

Questo è utilissimo quando scriviamo JS!



Tab Console

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. On the left, a sidebar displays message counts: 1 message (with an exclamation mark icon), 1 user message (with a person icon), 0 errors (with a red X icon), 0 warnings (with a yellow warning sign icon), 1 info (with a blue info icon), and 0 verbose (with a gear icon). The main console area shows the following output:

```
x Expression  
not available  
Sono una prova! index.html:12
```

At the bottom, a navigation bar includes links for 'Console', 'What's New', 'Search', 'Remote devices', and 'Request blocking'.

Ci fornisce anche l'output delle istruzioni di console.log (e tutta la famiglia).



Tab Console

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. On the left, a sidebar displays message counts: 2 messages, 2 user messages, 0 errors, 0 warnings, 2 info, and 0 verbose. The main console area shows the following log entries:

- x Expression
not available
- Sono una prova!
- > `console.log('Scrivo direttamente dalla console');`
Scrivo direttamente dalla console VM1161:1
- < undefined
- > |

At the bottom, tabs for 'Console', 'What's New', 'Search', 'Remote devices', and 'Request blocking' are visible.

Attraverso la console possiamo anche interagire con il JS e/o inserire altro codice.

Utilissimo per stare eventuali comportamenti "strani" del sito.



Tab Network

Name	Status	Type	Initiator	Size	Waterfall
index.html	Fini...	doc...	Other	337 B	1

Il tab network contiene tutte le chiamate che il browser fa per recuperare le risorse della pagina.

Ci fornisce informazioni sul loro esito e possiamo ispezionarle.



Tab Network

Name	Status	Type	Initiator	Size	T	Waterfall
index.html	Finished	document	Other	387 B	4.	
non_esiste.jpg	404	text/html	index.html	0 B	7.	

2 requests | 387 B transferred | 387 B resources | Finish: 132 ms | DOMContentLoaded: 24 ms | Load: 136 ms

Console What's New Search Remote devices Request blocking

Se le richieste ottengono come risposta un codice HTTP differente da 2XX (status code di successo) vengono mostrare in rosso.

Per queste richieste è utile ispezionare il motivo per il quale sono andate in errore.



Tab Network

The screenshot shows the Network tab in the Chrome DevTools. A request for 'non_esiste.jpg' has failed, resulting in a 404 Not Found error. The Headers section shows the following details:

- Request URL: http://google.com/non_esiste.jpg
- Request Method: GET
- Status Code: 404 Not Found
- Remote Address: 216.58.205.78:80
- Referrer Policy: no-referrer-when-downgrade

The Response Headers section includes:

- Content-Length: 1575
- Content-Type: text/html; charset=UTF-8
- Date: Sun, 08 Dec 2019 15:24:49 GMT
- Referrer-Policy: no-referrer

The Request Headers section lists:

- Accept: image/webp,image/apng,image/*,*/*;q=0.8
- Accept-Encoding: gzip, deflate
- Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.

At the bottom, the status bar indicates: 2 requests | 387 B transferred | 387 B resources | Finish: 132 ms | DOMCo.

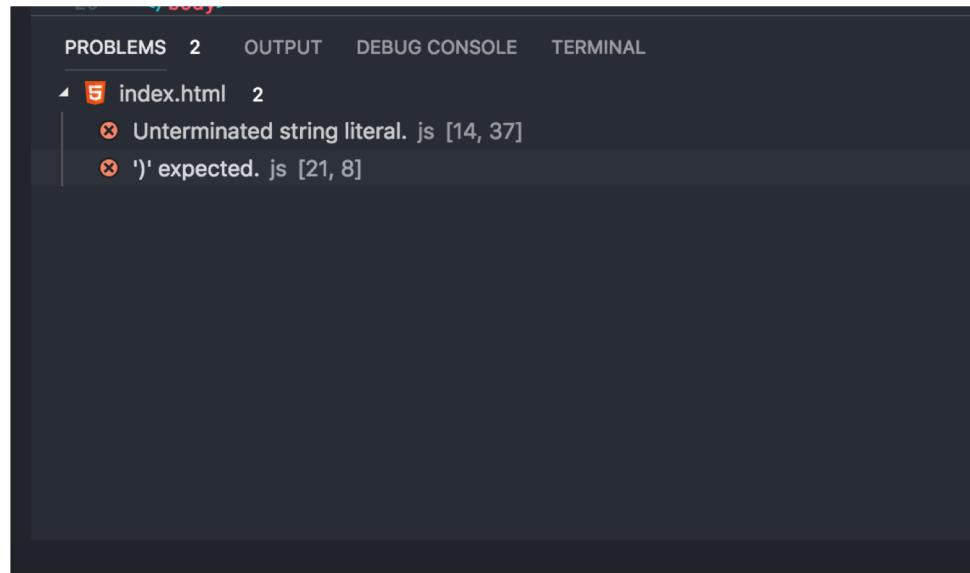
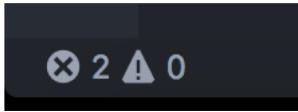
Se le richieste ottengono come risposta un codice HTTP differente da 2XX (status code di successo) vengono mostrate in rosso.

Per queste richieste è utile ispezionare il motivo per il quale sono andate in errore.



VSCode per JS

VSCode ci fornisce un utilissimo "grammar" check del codice JS scritto.





Estensioni utili per VSCode

Per lo sviluppo di JS ci sono alcune estensioni molto utili che ci aiutano a non commettere gli errori più comuni:

Bracket Pair Colorizer 2

Evidenzia le parentesi in modo da evitare di dimenticare quella di apertura e/o chiusura.

Beautify

Corregge l'indentazione di HTML/CSS e JS.

3

Esercizio 1



Analizzare homepage

Provate ad analizzare l'homepage di questi siti:

- Index.co
- Medium.com (aprire un articolo a caso)
- Google.com
- Amazon.com
- Netflix.com
- Twitter.com
- Facebook.com

Provate a rispondere:

- Quante richieste esterne fanno?
- Quanto pesano?
- Quanti file CSS hanno?
- Hanno errori/avvisi in console?

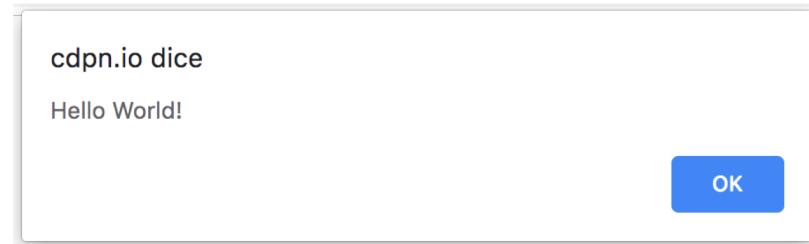
4

Sintassi di base



Esempio JS

```
<script>  
    alert('Hello World!');  
</script>
```



alert è un comando che permette di visualizzare una finestra popup nel browser per mostrare delle informazioni.

(Non ha niente a che fare con gli alert di bootstrap)



Primo esempio

```
<script>  
    alert('Hello World!');  
</script>
```



alert è un comando che permette di visualizzare una finestra popup nel browser per mostrare delle informazioni.

(Non ha niente a che fare con gli alert di bootstrap)



Scrivere nella pagina

```
<script>  
    document.write('<p>Hello World!</p>');  
</script>
```

Hello World!

document.write(..); ci consente di scrivere del contenuto nella pagina.

Nella quotidianità non è molto usata perché "non è abbastanza potente" come altre soluzioni (jQuery)



Scrivere nella pagina

```
<script>  
    document.write('<p>Hello World!</p>');  
</script>
```

Hello World!

Se visualizziamo il sorgente della pagina non troviamo il contenuto che abbiamo scritto.

Il contenuto è scritto dinamicamente dal browser all'interpretazione del codice JS.



Scrivere nella pagina

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Javascript test</title>
8 </head>
9 <body>
10  <script>
11    document.write( '<p>Hello World!</p>' );
12  </script>
13 </body>
14 </html>
```



Commenti

In JS esistono due tipi di commento:

- Commenti di una sola riga
- Commenti multiriga

```
<script>
    // Singola riga
</script>
```

```
<script>
    /* Questo commento
       e' composto da
       più righe */
</script>
```



Commenti a riga singola

```
<script>  
    // Singola riga  
</script>
```

- I commenti a riga singola si formano preponendo `//` prima del commento.
- Tutto ciò che si trova a destra di `//` fino alla fine della riga è considerato un commento.
- Si usano di solito per spiegazioni brevi o per annotazioni (brevi) sul funzionamento del codice.



Commenti multi riga

```
<script>
    /* Questo commento
       e' composto da
       più righe */
</script>
```

- I commenti multi riga si aprono scrivendo /* e si chiudono con */
- Tutto ciò che è compreso tra /* e */ è considerato commento e può occupare più linee.
- Il commento a riga singola termina automaticamente quando finisce la riga
- Si usano di solito per spiegazioni molto lunghe



Commentare il codice

Quando si programma esistono delle situazioni in cui è necessario "disabilitare" parte del codice per capire l'origine un bug.

Un modo molto semplice e usatissimo è commentare il codice problematico.

```
<script>
  /*
    console.log('Questo non viene eseguito');
  */
</script>
```



Documentare il codice

```
8 // Dear programmer:  
9 // When I wrote this code, only god and  
10 // I knew how it worked.  
11 // Now, only god knows it!  
12 //  
13 // Therefore, if you are trying to optimize  
14 // this routine and it fails (most surely),  
15 // please increase this counter as a  
16 // warning for the next person:  
17 //  
18 // total_hours_wasted_here = 254  
19 //  
20
```

I commenti hanno la funzione di spiegare il funzionamento del codice.

Meglio un commento in più che un commento in meno!



Punto e virgola

Javascript non obbliga la separazione delle istruzioni attraverso il punto e virgola ":".

E' sempre buona regola metterlo comunque per separare le istruzioni e suddividere meglio il codice.

```
<script>  
    console.log('Senza ;')  
</script>
```

```
<script>  
    console.log('Con ;');  
</script>
```



Variabili

Una variabile deve essere pensata come un contenitore di qualcosa (un numero, una stringa, etc).

Sono usate per contenere un valore "temporaneo" che può variare durante l'esecuzione del codice (da cui il nome variabile)

Le variabili sono identificate da un nome che permette di richiamarle nel codice.



Dare un nome alle variabili

Un nome valido di una variabile:

- Non **deve** coincidere con una parola chiave del linguaggio
- Non può iniziare con un numero
- Non può contenere caratteri speciali (spazi, lettere accentate, il trattino, etc)
- Può contenere un underscore
- Può contenere o iniziare con il simbolo del dollaro (\$)



Le costanti in JS

Javascript (fino alle versione ES5) non conosce il concetto di costante, ovvero di un valore definito una sola volta e che non può (e non deve) variare durante l'esecuzione del codice.

Ad esempio le costante matematiche non devono variare (pi greco)

Convenzionalmente si è deciso che le costante sono semplicemente variabili scritte tutte in maiuscolo con le parole eventualmente separate da _ (undescore). Es. PIGRECO, LA_MIA_COSTANTE



Le costanti in JS

```
var PIGRECO = 3.14; // Questa è una costante per convenzione
```

Dalla versione ES6 è stato introdotto la possibilità di dichiarare costanti usando la parola chiave **const**. Naturalmente la convenzione di usare nomi maiuscoli, eventualmente separati da _ (underscore) deve essere comunque rispettata: aumenta la leggibilità.

```
const PIGRECO = 3.14; // Questa è una costante per ES6
```



Tutti i nomi sono "buoni nomi"?

Il codice che scrivete deve essere autoesPLICATIVO

Il codice si dice essere autoesPLICATIVO se è di facile comprensione e i nomi di variabili, funzioni etc rispecchiano esattamente quello che contengono/fanno.

Questo è di fondamentale importanza perché il codice deve essere letto da altri programmatore e più facile capire cosa fa e più è facile apportare modifice/migliorare/correggere problemi.



Tutti i nomi sono "buoni nomi"?

```
var a = 31; // Cosa significa??
```

```
var giorni = 31; // Già meglio, stiamo parlando di giorni
```

```
var numeroDiGiorniInDicembre = 31; // Molto meglio
```



I tipi di dato: stringe

```
<script>  
    var string1 = 'abcd';  
    var string2 = "abcd";  
    var empty = '';  
</script>
```

Le stringe sono sequenze di caratteri.

Le stringe sono delimitate da '' oppure da ''. Sono validi entrambi e non fanno differenza.

Scegliete pure quello che vi piace ma state **coerenti**: fare le cose sempre nello stesso modo!



I tipi di dato: stringhe

Le stringhe possono essere concatenate: ovvero "sommare" due stringhe per formarne una composta dalla giustapposizione della prima e della seconda.

```
<script>
    var string1 = 'abcd';
    var string2 = "efgh";
    var string3 = string1 + string2;
    console.log(string3);
</script>
```

The screenshot shows a browser's developer tools console window. The input field contains the expression `string3` followed by a plus sign. The output panel displays the result of the concatenation: `abcdefgh`. Above the output, there is a message: `x Expression not available`.

```
x Expression
not available
abcdefgh
>
```



I tipi di dato: numeri

```
<script>  
    var numero1 = 6;  
    var numero2 = 3.14;  
</script>
```

Javascript non fa differenza tra numero intero (es 6) e i numeri decimali (es 3.14).

Si usa la notazione americana, quindi il punto al posto della virgola.

Naturalmente sono supportate tutte le principali operazioni sui numeri.



I tipi di dato: boolean

```
<script>  
    var vero = true;  
    var falso = false;  
</script>
```

Javascript possiede valori booleani, ovvero valori che assumono solamente uno di due valori: TRUE o FALSE.

Sono utilissimo soprattutto nella logica condizionale: if, while, etc dove si decidere di eseguire del codice solamente se si verifica una determina condizione.



I tipi di dato: array

```
<script>  
    var giorniDellaSettimana = [  
        "lunedì",  
        "martedì",  
        "mercoledì",  
        "giovedì",  
        "venerdì",  
        "sabato",  
        "domenica"  
    ];  
</script>
```

Gli array sono un insieme numerato di variabili tutte dello stesso tipo.

E' possibile accedere ad uno specifico elemento attraverso il suo indice (gli indici partono da 0)



I tipi di dato: array

```
var giorniDellaSettimana = [  
    "lunedì",  
    "martedì",  
    "mercoledì",  
    "giovedì",  
    "venerdì",  
    "sabato",  
    "domenica"  
];  
console.log(giorniDellaSettimana[0]);  
console.log(giorniDellaSettimana[6]);
```

x Expression
not available

lunedì
domenica

>



Gli operatori matematici

Operatore	Nome
+	addizione
-	sottrazione
/	divisione
*	moltiplicazione
%	modulo o resto



Gli operatori matematici unari

Operatore	Nome
-	negazione
++	incremento
--	decremento

```
var a = 10;  
console.log(a++);  
console.log(a);  
console.log(++a);  
console.log(a);
```

x Expression
not available

10
11
12
12
>



Gli operatori relazionali

Operatore	Nome
<	minore
<=	minore o uguale
>	maggior
>=	maggior o uguale
==	uguale
!=	diverso
==	strettamente uguale
!==	strettamente diverso



Gli operatori logici

Operatore	Nome
&&	and
	or
!	not



Gli operatori di assegnamento

Forma compatta	Scrittura equivalente
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x %= y$	$x = x \% y$



Istruzione alert

```
<script>  
    alert('sono una prova');  
</script>
```



L'istruzione **alert** ci permette di far apparire un box con in cui mostrare un messaggio di attenzione all'utente. Il box contiene un bottone OK per chiudere la finestra. Sono sconsigliate in un sito ed è meglio usare qualcosa di più carino tipo i modal di bootstrap.



Istruzione console.log

```
<script>  
    console.log('test');  
</script>
```

x Expression
not available
test
>

L'istruzione console.log ci permette di mostrare in console delle informazioni. E' perfetta nello sviluppo per stampare a console delle variabili con contenuto "ignoto", frutto di qualche operazione.

Quando il sito va in produzione la console deve essere "pulita", ovvero non devono rimanere nel codice istruzioni di console.log.



Istruzione console.log

In caso di problemi con il codice può essere usata per capire a che punto il codice si blocca.

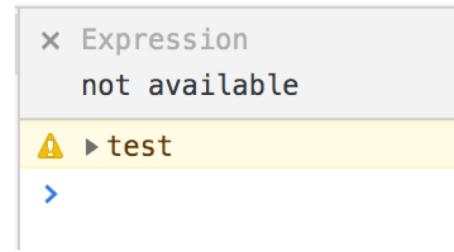
```
<script>
    var a = 10;
    console.log('a inizializzato');
    var b = 11;
    console.log('b inizializzato');
    var c = nonEsito(a, b);
    console.log('eseguita la funzione nonEsisto');
</script>
```

```
x Expression
not available
a inizializzato
b inizializzato
✖ ▶ Uncaught ReferenceError: nonEsito is not defined
at index.html:18
>
```



Istruzione console.warn

```
<script>  
    console.warn('test');  
</script>
```



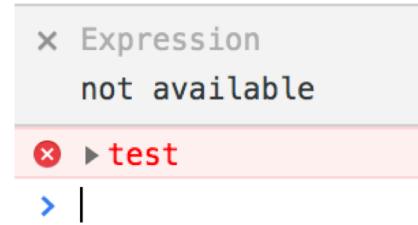
Identica come funzionamento a console.log, ma mostra un messaggio di attenzione.

Deve essere usata per indicare all'utente che qualcosa potrebbe funzionare a dovere ma non è detto. E' un messaggio di attenzione.



Istruzione console.error

```
<script>  
    console.error('test');  
</script>
```

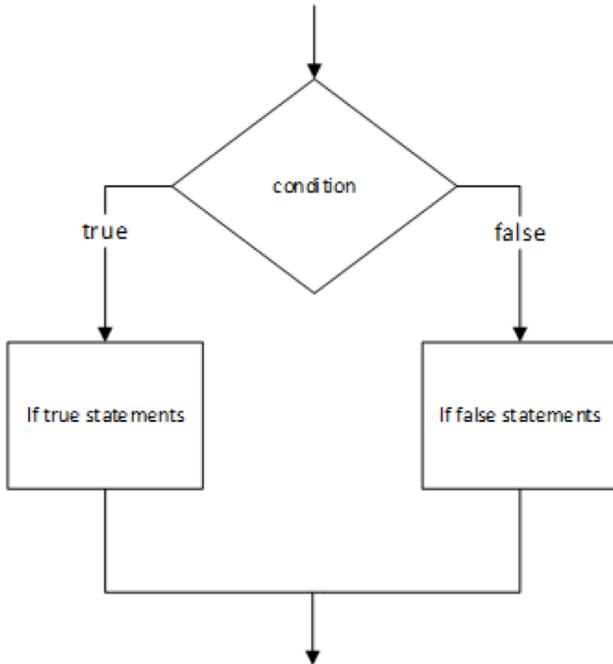


Funziona esattamente come console.log e console.warn ma mostra un messaggio di errore.

Può essere usata per avvertire l'utente che c'è stato qualche problema e che quindi il sito non funzionerà come voluto.



Controllo di flusso: if



Quando scriviamo il codice capita spesso di dover eseguire operazioni differenti se si verifica una determinata condizione.



Controllo di flusso: if

Being a Programmer

Mom said: "Please go to the shop and buy 1 bottle of milk. If they have eggs, bring 6"

I came back with 6 bottle of milk.



She said: "Why the hell did you buy 6 bottles of milk?"

I said: "BECAUSE THEY HAD EGGS"



Controllo di flusso: if

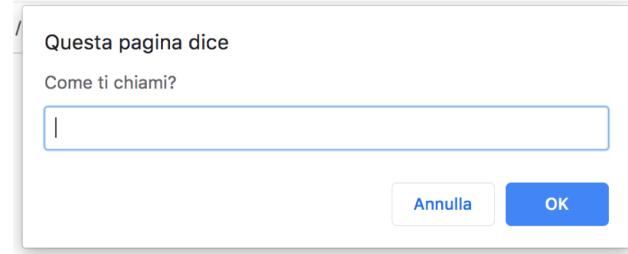
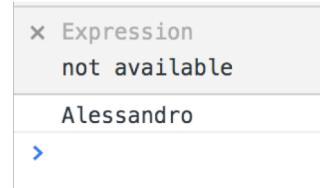
```
if (codizione1) {  
    // Codice se condizione è vera  
} [else if (codizione2) {  
    // Codice se condizione1 falsa e condizione2 vera  
} [else {  
    // Entrambe le condizioni sono false  
}]]
```



Prompt

Prompt permette di richiedere all'utente l'immissione di alcune informazioni attraverso una finestra di dialogo.

```
<script>
    var name = prompt('Come ti chiami?');
    console.log(name);
</script>
```



5

Esercizio 2



Costruire una calcolatrice

Creare una calcolatrice dove all'utente è richiesto l'inserimento di un operando, dell'operatore e del secondo operando e mostrare nella pagina il risultato dentro un div di colore rosso se il risultato è negativo e verde se positivo.

Se l'operazione scelta è una divisione e il secondo operando è 0 mostrare un alert con scritto che il non è possibile fare divisioni per 0.

5

Esercizio 3



Controllo di flusso

Modificare l'esercizio precedente in modo che uno dei due operatori è nullo mostrare un alert indicante l'errore.



Iterazioni

Nella programmazione a volte è necessario ripetere del codice per un predeterminato numero di volte oppure finchè una condizione non diventa falsa.

Javascript (così come gli altri linguaggi) possiede due costrutti:
for e **while**

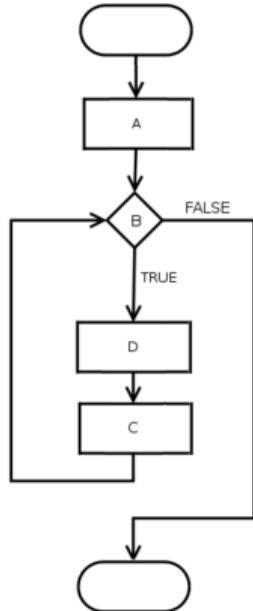
for: ripete il codice per un determinato numero di volte

while: ripete il codice finchè una condizione non diventa false



Iterazioni: for

```
for(A;B;C)  
D;
```



A è un'istruzione di inizializzazione: inizializziamo una variabile contatore al suo valore minimo.

B è una condizione: finchè è valida esegui il codice D.

C è un'istruzione di incremento: incremento la variabile contatore



Iterazioni: for

```
<script>
    for (var i = 0; i < 10; i++) {
        console.log(i);
    }
</script>
```

x Expression
not available

0
1
2
3
4
5
6
7
8
9
> |



Iterazioni: for

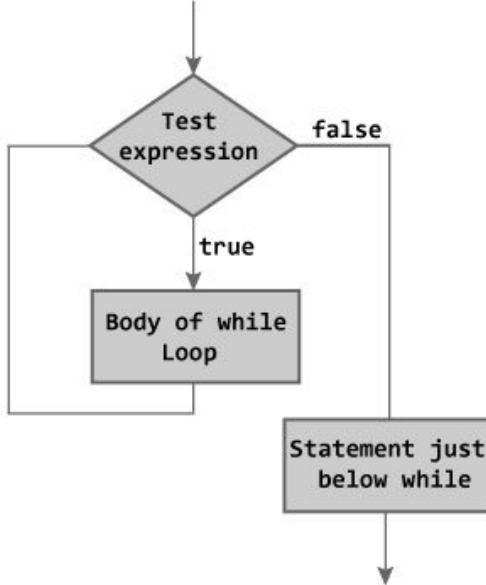
```
for (init; condizione; incremento) {  
    // Codice da ripetere  
}
```

Da notare che init, condizione e incremento sono separati da ;

Questo perché tutte e 3 sono istruzioni



Iterazioni: while



Test expression è la nostra condizione.

"Funziona similmente ad un if", ovvero funchè la condizione è vera, allora il codice viene ripetuto.

Figure: Flowchart of while Loop



Iterazioni: while

```
<script>
    var i = 0;
    while (i < 10) {
        console.log(i);
        i++;
    }
</script>
```

x Expression
not available

0
1
2
3
4
5
6
7
8
9
> |



Iterazioni: while

```
while (condizione) {  
    // Codice da ripetere  
}
```



for vs while

I costrutti per fare i cicli sono completamente equivalenti.

Il fatto che ne esistano diversi è solamente per aiutare il programmatore a scrivere codice più leggibile.

```
<script>
  for (var i = 0; i < 10; i++) {
    console.log(i);
  }
</script>
```

```
<script>
  var i = 0;
  while (i < 10) {
    console.log(i);
    i++;
  }
</script>
```



onclick

E' possibile eseguire del codice JS in risposta ad un evento.

Il più semplice è l'evento onclick, dove è possibile eseguire una funzione in risposta al click dell'evento.

```
<button onclick="eseguiClick()">Test</button>  
<script>  
    function eseguiClick() {  
        alert('Hai premuto il tasto');  
    }  
</script>
```



Modificare il contenuto

JS permette di identificare un elemento attraverso il suo id e modificare il suo contenuto.

```
<button id="test1" onclick="eseguiClick()">Premi</button>
<script>
    function eseguiClick() {
        var el = document.getElementById('test1');
        el.innerHTML = "Sono stato premuto";
    }
</script>
```

Premi

Sono stato premuto



Aggiungere/Rimuovere classi

JS permette di aggiungere una o più classi di un elemento.

```
<button id="test1" onclick="eseguiClick()">Premi</button>
<script>
    var el = document.getElementById('test1');
    el.classList.add("my-class");
</script>
```

```
<button id="test1" onclick="eseguiClick()" class="my-class">Premi</button>
```



Aggiungere/Rimuovere classi

JS permette di rimuovere una o più classi di un elemento.

```
<button id="test1" onclick="eseguiClick()" class="my-class">Premi</button>
<script>
  var el = document.getElementById('test1');
  el.classList.remove("my-class");
</script>
```

```
<button id="test1" onclick="eseguiClick()" class>Premi</button> == $0
```



Ottenerne il valore di un input

JS permette di ottenere il contenuto di un elemento di input

```
<input id="test1" value="test123">  
<script>  
    var el = document.getElementById('test1');  
    var value = el.value;  
    console.log(value);  
</script>
```

x Expression
not available
test123

>



Modificare il valore di un input

JS permette di ottenere il contenuto di un elemento di input

```
<input id="test1" value="">  
<script>  
  var el = document.getElementById('test1');  
  el.value = '1234';  
</script>
```

1234

6

Esercizio 4



Calcolatrice 2

Creare una calcolatrice con 4 pulsanti, uno per operazione e due input per inserire gli operandi.

Creare un div per scrivere il risultato dell'operazione.

Se il risultato è positivo mettere il testo in verde, se è negativo in rosso.

7

Esercizio 5

Traveler

Rifare la pagina usando bootstrap e suoi componenti.

Deve essere responsive!

https://github.com/howtosecondo/frontend-padova/raw/master/2019_11_28/es1.zip

Travel
Travel diaries

HOME BLOG RESUME CONTACT

Reflections on my holiday in the United States...

I'M A GREAT TRAVELLER

Placeholder text (Lorem ipsum) for the travel blog post.

ABOUT THE AUTHOR

Placeholder text (Lorem ipsum) about the author, featuring a photo of a cat.





©2015 Ronnie Filyaw WHOMPCOMIC.COM





Thanks!

Any questions?

Per qualsiasi domanda contattatemi:
alessandro.pasqualini.1105@gmail.com