Lo sviluppo delle competenze di 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

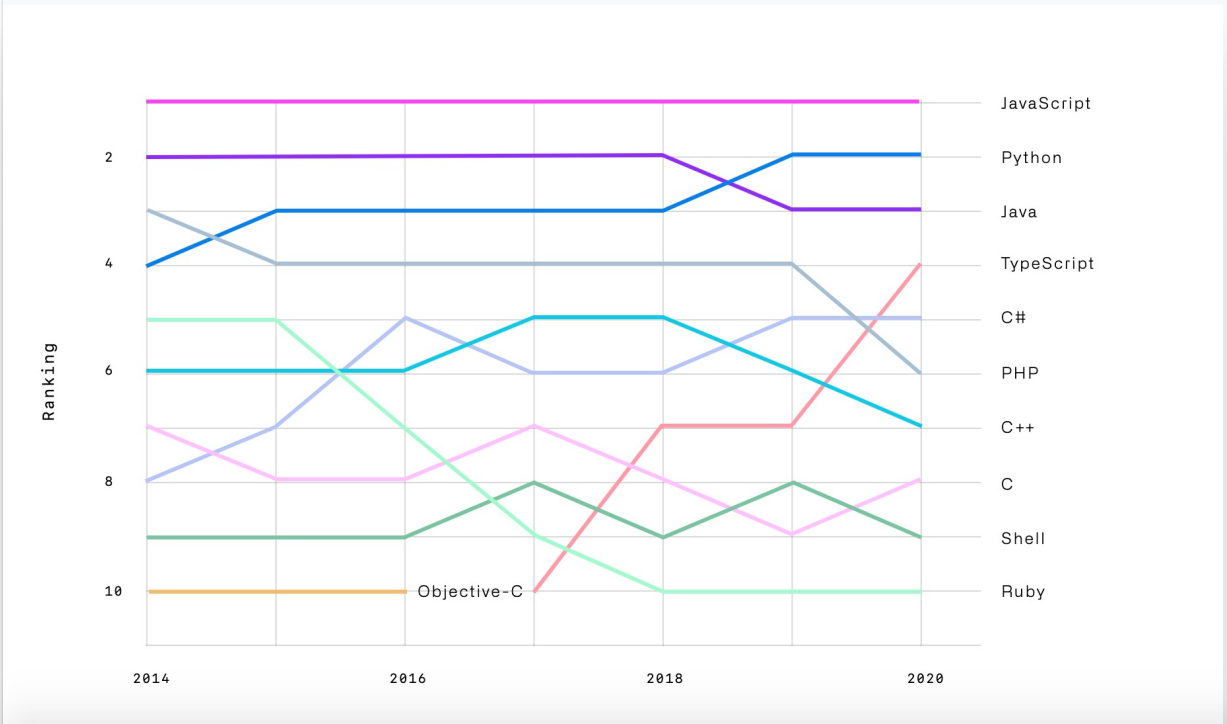
Cos'è Javascript?

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 nate per il web per realizzare la cosiddetta business logic dell'applicazione, ovvero il backend.

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 proprità, accedere al CSS, accedere al contenuto, etc

Modificare un elemento: modificare il contenuto di un elemento, creare elementi "on the fly", cambiare le sue proprità, 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 contento. (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 includo 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 attorno ad 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 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.

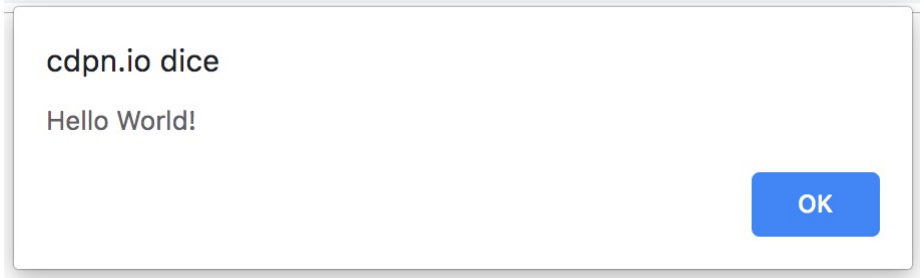
Questa regola generale non si applica ad alcuni script particolari (ad esempio il tracciamento dell'utente o il blocco dei cookies) che richiedono di essere inizializzati prima del contenuto (e del resto del codice JS)

2

Sintassi di base

Esempio JS

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

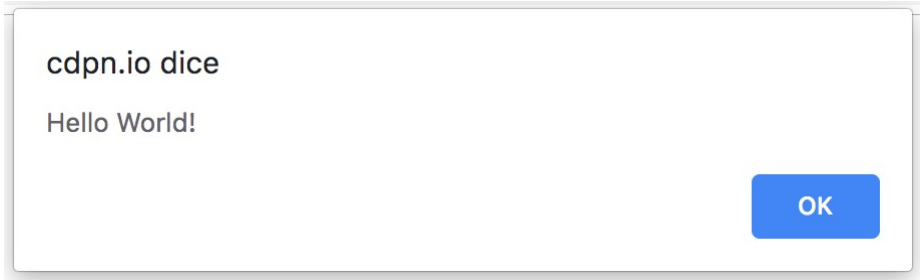


alert è un comando che permette di visualizzare una finestra popup nel browser per mostrare delle informazioni.

(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

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Javascript test</title>
</head>
<body>
    <script>
        document.write('<p>Hello World!</p>');
    </script>
</body>
</html>
```

Commenti

In JS esistono due tipi di commento:

- Commenti di una sola riga
- Commenti multiriga

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

```
<script>
    /* Questo commmento
       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 commmento
       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

```
// Dear programmer:
// When I wrote this code, only god and
// I knew how it worked.
// Now, only god knows it!
//
// Therefore, if you are trying to optimize
// this routine and it fails (most surely),
// please increase this counter as a
// warning for the next person:
//
// total_hours_wasted_here = 254
//

```

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 ";".

Non c'è una regola, **l'importante è essere consistenti!** Se si decide di metterlo allora è un errore non metterlo, viceversa se si decide di non metterlo è un errore metterlo!

```
<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 costanti 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 introdotta la possibilità di dichiarate costanti usando la parola chiave **const.** Naturalmente la convenzione di usare nomi maiuscoli, eventualmente seprarati da _ (undescore) deve essere comunque rispettata: aumenta la leggibilità del codice.

```
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 programmatori e più facile capire cosa fa e più è facile apportare modifice/migliorire/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; // Ah, adesso si capisce!
```

I tipi di dato: stringe

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

Le stringe sono sequenze di caratteri.

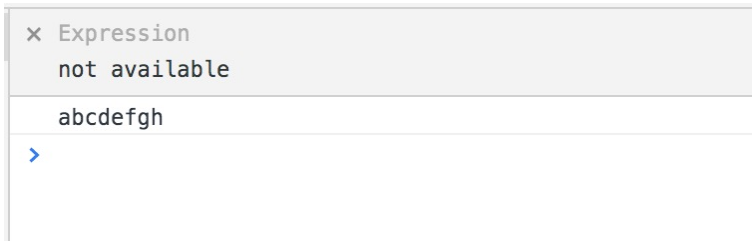
Le stringe sono delimitate da **" "** oppure da **' '**. Sono validi entrambi e non fanno differenza.

Scegliete pure quello che vi piace ma siate **consistenti: fate 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>
```



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: bolean

```
<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

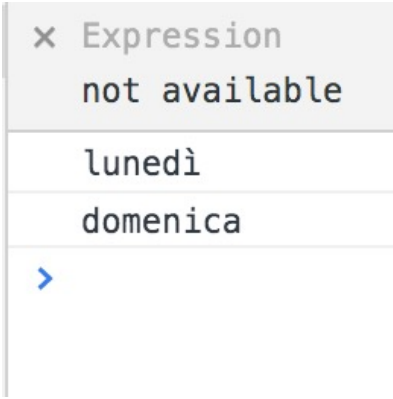
```
<script>
    var giorniDellaSettimana = [
        "lunedì",
        "martedì",
        "mercoledì",
        "giovedì",
        "venerdì",
        "sabato",
        "domenica"
];
</script>
```

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]);
```



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);
```

```
× Expression not available
10
11
12
12
>
```

Gli operatori relazionali

Operatore	Nome
`<`	minore
`<=`	minore o uguale
`>`	maggiore
`>=`	maggiore 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>
```

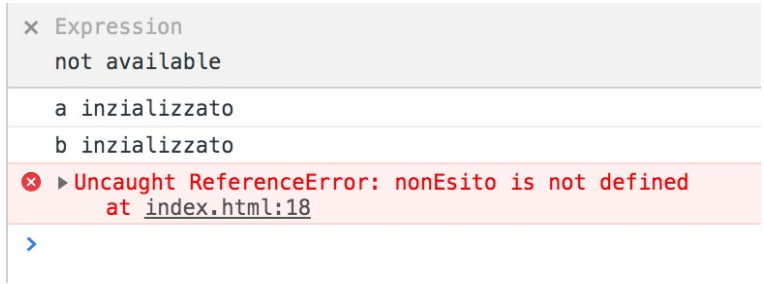
L'istruzione console.log ci permette di mostrare in console delle informazioni. E' perfetta nello sviluppo per stampare a console delle variabili con contenuto "ignoto", frutto di qualche operazione.

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

Istruzione console.log

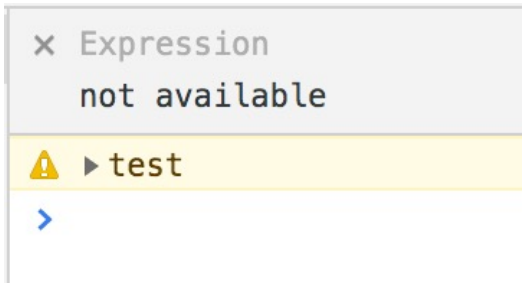
In caso di problemi con il codice può essere usata per capire a che punto il codice si blocca.

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



Istruzione console.warn

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

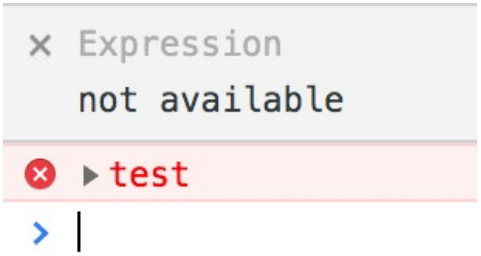


Identica come funzionamento a console.log, ma mostra un messaggio di attenzione.

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

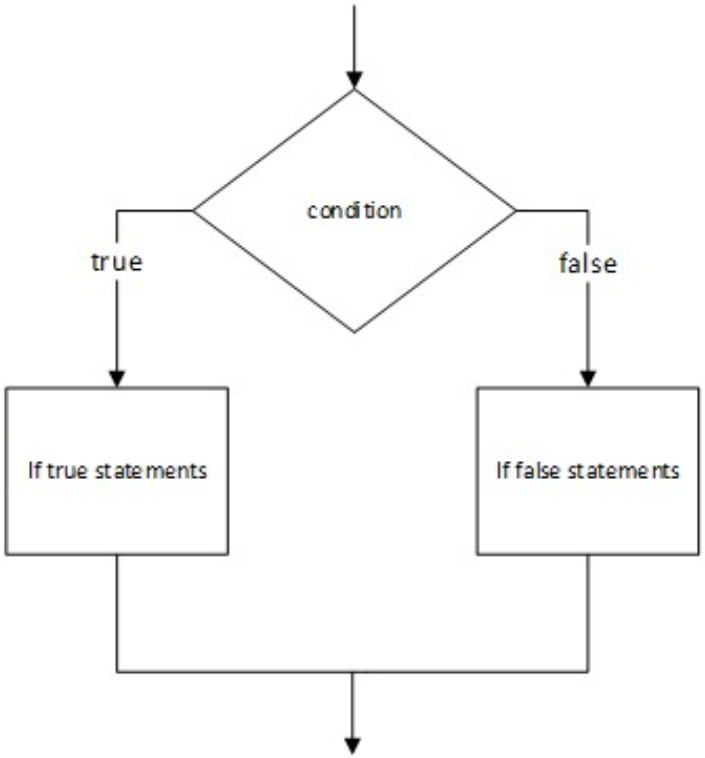
```
<script>
    console.error('test');
</script>
```



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

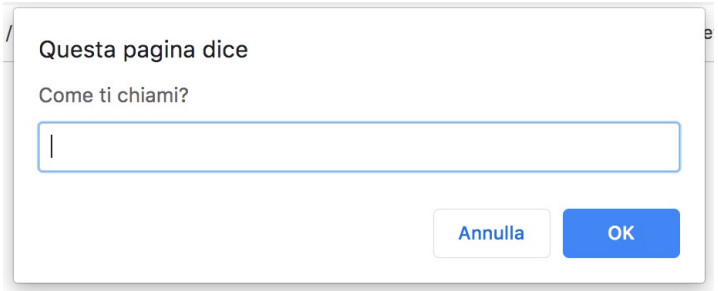
Controllo di flusso: if

```
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>
```



3

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.

4

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.

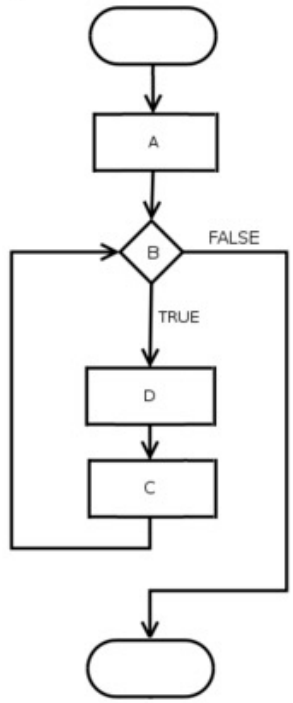
Javascript (così come gli altri linguaggi) possiede due costrutti: **for** e **while**

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: inizializiamo 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>
```

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

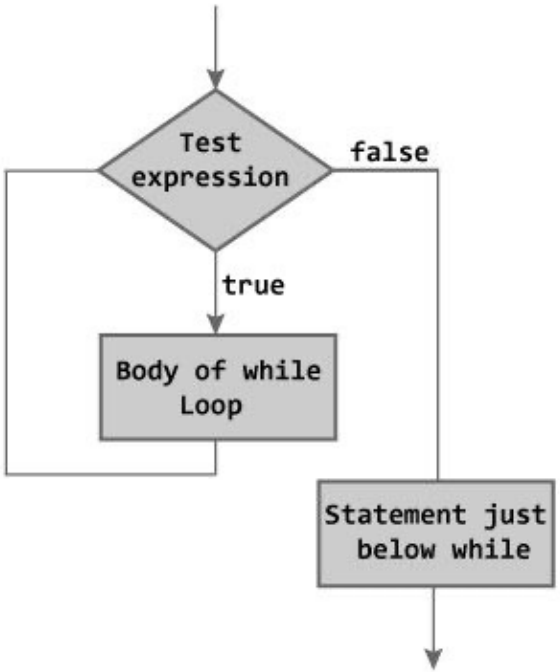


Figure: Flowchart of while Loop

Test expression è la nostra condizione.

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

Iterazioni: while

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

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 indentificare 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>
```

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
```

Ottenere 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>
```

```
× 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

5

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.


SINCE YOU'RE COMING OVER, CAN YOU PICK ME UP A LOAF OF MILK, A DOZEN BREAD AND A GALLON OF EGGS?
AH, HE KNOWS WHAT I MEANT.
HA, HA.
©2015 Ronnie Filyaw WHOMPCOMIC.COM
YOU DID IT.
I DON'T KNOW HOW, BUT YOU DID IT.
EGGS
1 GAL
MILK
LOAF