

**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

Ripasso 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



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.

Di norma il codice JS va inserito come ultimo elemento del body.

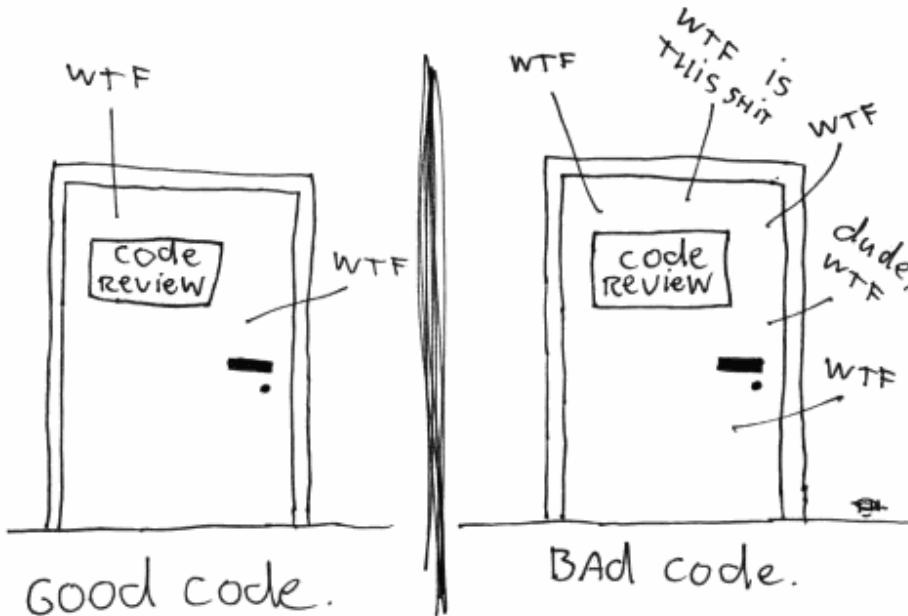
Naturalmente esistono eccezioni ma si valuta caso per caso.

2

Indentazione e coding style



The ONLY VALID MEASUREMENT OF Code QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift

“

You know you are working on clean code when each routine you read turns out to be pretty much what you expected. You can call it beautiful code when the code also makes it look like the language was made for the problem.

- Ward Cunningham



Perchè "clean code"?

La principale attività del programmatore è leggere e comprendere codice.

E' di fondamentale importanza scrivere codice comprensibili e facile da leggere. Più è facile da leggere meno tempo "perdiamo" nel comprendere il codice e più tempo possiamo dedicare ad attività "più produttive".



Coding style

```
var x=10;if(x>10)for(var i=0;i<10;i++)console.log(i);else  
console.log('boh');
```



Coding style

```
var x=10;  
if(x>10) {  
    for(var i=0;i<10;i++) {  
        console.log(i);  
    }  
}else{  
    console.log('boh');  
}
```



Coding style

```
var x = 10;  
  
if (x > 10) {  
    for (var i = 0; i < 10; i++) {  
        console.log(i);  
    }  
} else {  
    console.log('boh');  
}
```



Documentare il codice

I commenti servono la funzione di documentazione del codice.

Il codice scritto bene ha bisogno di commenti che descrivono il comportamento di parti complicate del codice.

I commenti inutili o che dicono cose "ovvie" sono peggio di non avere commenti.



Documentare il codice

I commenti servono la funzione di documentazione del codice.

Il codice scritto bene ha bisogno di commenti che descrivono il comportamento di parti complicate del codice.

I commenti inutili o che dicono cose "ovvie" sono peggio di non avere commenti.



Nomi significativi

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.

E' molto probabile che il vostro codice venga letto da altri programmatori (anche voi) dopo qualche tempo e l'unico modo di ricostruire il funzionamento è che il codice sia semplice, pulito e ordinato.

3

Ripasso veloce



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
```



I tipi di dato

Ogni variabile possiede un tipo di dato, ovvero:

- L'insieme dei valori permessi
- Le operazioni che si possono effettuare

Javascript viene detto loosely typed, ovvero non è necessario specificare a priori il tipo della variabile ma viene "assegnato" automaticamente da Javascript.

Questo è fonte di molti errori ed è sempre bene prestare particolare attenzione a cosa stiamo cercando di fare!



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. L'operatore di concatenazione è il +

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

```
x Expression
not available
abcdefg
>
```



I tipi di dato: numeri

```
var numero1 = 6;  
var numero2 = 3.14;
```

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

```
var vero = true;  
var falso = false;
```

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

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

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
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x %= y$	$x = x \% y$



I tipi di dato

```
var n1 = 1;  
var n2 = '2';  
var n3 = n1 + n2;  
console.log(n3);
```

```
x Expression  
not available  
12  
>
```

```
var n1 = 1;  
var n2 = 2;  
var n3 = n1 + n2;  
console.log(n3);
```

```
x Expression  
not available  
3  
>
```



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 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.



Controllo di flusso: if

```
var x = 10;  
var y = 9;  
  
if (x > y) {  
    console.log('x > y');  
} else {  
    console.log('x < y');  
}
```





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: while

```
var x = 10;  
var y = 7;  
while (x > y) {  
    console.log(x);  
    x--;  
}
```

x Expression
not available

10

9

8

> |



Istruzione switch

```
switch (variabile) {  
    case '1':  
        console.log('1');  
        break;  
    case '2':  
        console.log('2');  
        break;  
  
    default:  
        console.log('default');  
}
```



Istruzione switch

```
switch (variabile) {  
    case '1':  
        console.log('1');  
        break;  
    case '2':  
        console.log('2');  
        break;  
  
    default:  
        console.log('default');  
}
```

```
if (variabile == '1') {  
    console.log('1');  
} else if (variabile == '2') {  
    console.log('2');  
} else {  
    console.log('default');  
}
```



Istruzione switch

Switch è un costrutto che permette di sostituire una cascata di if e di migliorare la leggibilità del codice.

E' meno potente del costrutto if e ogni **case** deve obbligatoriamente essere "interrotto" da un istruzione **break**;



Istruzione switch (con break!)

```
var variabile = '1';

switch (variabile) {
    case '1':
        console.log('1');
        break;

    case '2':
        console.log('2');
        break;

    default:
        console.log('default');
}
```

x Expression
not available

1

> |



Istruzione switch (senza break!)

```
var variabile = '1';

switch (variabile) {
    case '1':
        console.log('1');

    case '2':
        console.log('2');

    default:
        console.log('default');
}
```

x Expression
not available

1
2
default

> |

4

Esercizio



Esercizio

Creare un ciclo che scorra i primi 10 numeri naturali (1, 2, 3...) e dire se sono pari o dispari usando solamente switch.

No if!

5

Le funzioni



Le funzioni

```
function isEven(number) {  
    if (number % 2 == 0) {  
        console.log('Pari');  
    } else {  
        console.log('Dispari');  
    }  
}  
isEven(2);  
isEven(3);
```

```
x Expression  
not available  
Pari  
Dispari  
> |
```



Le funzioni

Le funzioni sono un modo per raggruppare insieme del codice in modo da poterlo rieseguire senza doverlo riscrivere.

Le funzioni derivano dal concetto matematico di funzione dove dati determinati parametri di ingresso ci vengono ritornati uno o più parametri "calcolati" dalla funzione.

Sono utilizzate anche per raggruppare logicamente il codice.



Le funzioni

Le funzioni in Javascript posseggono (quasi sempre) un nome e dei parametri di ingresso e restituiscono un valore.

Se la funzione non ha un nome si chiama **funzione anonima**.

Le funzioni anonymous sono molto usate per raggruppare il codice e le useremo con jQuery.



Nomi delle funzioni

I nomi delle funzioni, così come i nomi delle variabili, devono essere univoci all'interno di un blocco di codice.

Per Javascript un blocco è il codice definito all'interno di una coppia di parentesi graffe.



Nomi delle funzioni

```
{  
  function a() {  
    console.log('x');  
  }  
  a();  
}  
  
{  
  function a() {  
    console.log('y');  
  }  
  a();  
}
```

x Expression
not available

x

y

>



Nomi delle funzioni

```
function a() {  
    console.log('x');  
}  
a();
```

```
function a() {  
    console.log('y');  
}  
a();
```



Nomi delle funzioni

```
function a() {  
    console.log('x');  
}  
a();  
  
function a() {  
    console.log('y');  
}  
a();
```

x Expression
not available

y

y



Parametri di ingresso

I parametri di ingresso non sono altro che delle variabili che vengono passate alla funzione.

Sono usate per passare dei valori necessari all'esecuzione del codice della funzione.

Possono essere in qualsiasi numero e di qualsiasi tipo e numero.



Parametri di ingresso

```
function test(p1, p2, p3) {  
    console.log(typeof p1);  
    console.log(typeof p2);  
    console.log(typeof p3);  
}
```

```
test(1,2,3);  
console.log('-----');  
test(1);
```

x Expression
not available

number
number
number

number
undefined
undefined

>



Parametri di ingresso

Se un parametro non viene passato durante l'invocazione della funzione, Javascript gli assegna il tipo speciale `undefined`

`Undefined` significa che la variabile/parametro non è definito.

Prestare attenzione a questi "casi particolari" e passare sempre tutti i parametri alla funzione. E' bad practise passare meno parametri a meno che non siano definiti dei valori di default.



Parametri di ingresso

```
function test(parametro = false) {  
    console.log(parametro);  
}  
  
test(true);  
test();
```

x Expression
not available

true

false

>



L'istruzione return

L'istruzione `return` è la funzione che permette alla funzione di ritornare un valore al termine della funzione.

Può ritornare un valore unico (gli array sono comunque un valore unico).

Può essere usata per interrompere l'esecuzione della funzione se usata da sola.



Le funzioni

```
function isOdd(number) {  
    if (number % 2 != 0) {  
        return true;  
    }  
  
    return false  
}  
  
console.log(isOdd(2));  
console.log(isOdd(3));
```

x Expression
not available

false
true
➤ |



L'istruzione return

Le funzioni ritornano sempre un valore, anche se non è specificata l'istruzione return oppure è vuota.

Questo valore è **undefined** ed è il valore di default di javascript.

Significa letteralmente **non definito**.



Variabili locali e variabili globali

Le variabili definite all'esterno delle funzioni sono chiamate variabili globali.

Le variabili definite all'interno delle funzioni sono chiamate variabili locali.

La differenza sostanziale è che le variabili globali esistono globalmente, ovvero sono accessibili anche all'interno della funzione, mentre quelle locali hanno vita solamente durante l'esecuzione della funzione. Quando la funzione termina vengono distrutte.



Variabili locali e variabili globali

```
function test(number) {  
    var a = number;  
}  
var b = 10;  
test(b);  
console.log(a);  
console.log(b);
```

The screenshot shows a browser's developer tools error panel. At the top, there is a message: "x Expression not available". Below it, a red error message is displayed: "✖ ▶ Uncaught ReferenceError: a is not defined at index.html:19". A blue arrow points downwards from the error message.



Scoping delle variabili

```
function test(number) {  
    var a = number;  
    console.log(a);  
}  
  
var a = 10;  
test(17);  
console.log(a);
```

x Expression
not available

17
10
> |



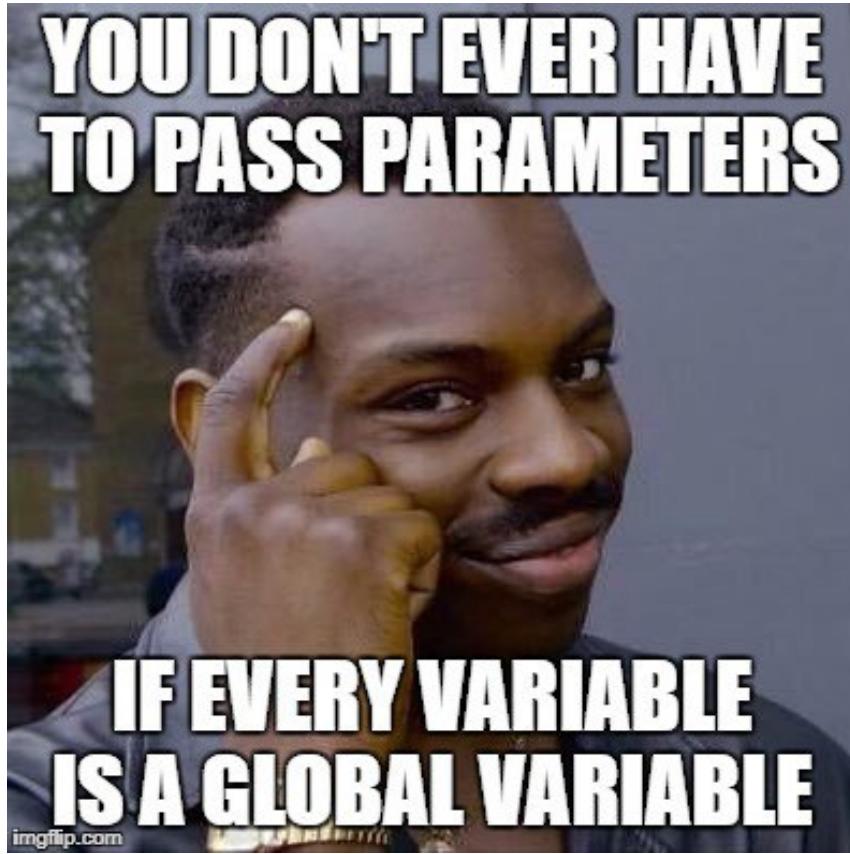
Scoping delle variabili

```
function test(number) {  
    a = number;  
    console.log(a);  
}  
var a = 10;  
test(17);  
console.log(a);
```

x Expression
not available

17
17

>



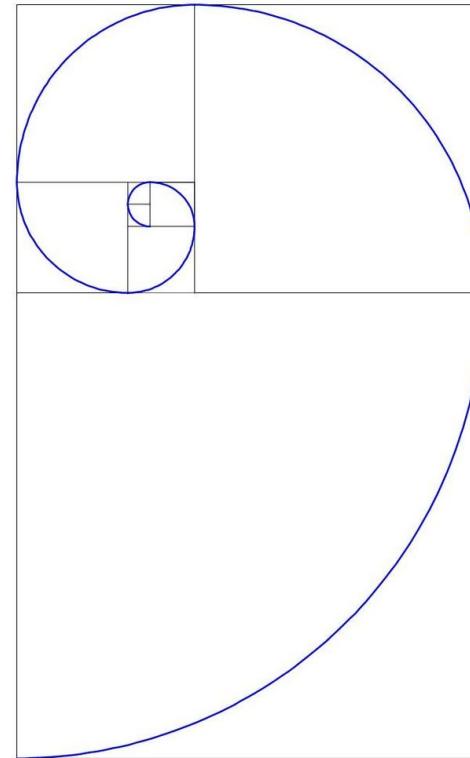


La sezione aurea

La sezione aurea è chiamata anche proporzione divina.

E' usatissima in edilizia e in tantissime altre discipline tra cui ... lo sviluppo di siti internet.

Quando dovete fare un rettangolo usate la sezione aurea come rapporto tra la base e l'altezza. Verrà sempre bene!





La sezione aurea

The image displays two web pages illustrating the use of the golden ratio in design:

Left Page (Twitter Profile):

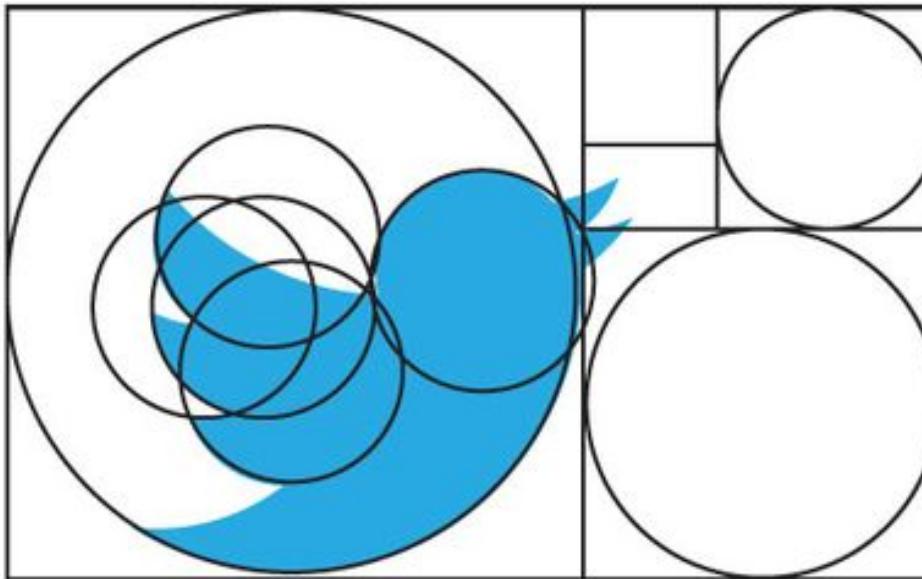
- The sidebar features a Fibonacci sequence diagram (a spiral composed of squares with side lengths corresponding to the numbers in the sequence).
- The top navigation bar has a red header bar.
- The footer includes a "Follow" button.

Right Page (Political Campaign Website):

- A large blue circle is drawn over the top half of the page, enclosing the main headline area.
- The main headline reads "PICK RICK." in large white letters.
- To the right of the headline, there's a section titled "SEND SHOCKWAVES THROUGH THE WHITE HOUSE." with a "DONATE NOW" button.
- Below the headline, there's a "JOIN the FIGHT" section with various calls to action like "EVEN", "VOLUNTEER", "E-MAIL", "SIGN-UP", "CONTRIBUTE", "SHOP", and "CALL FROM HOME".
- A golden ratio grid is overlaid on the bottom right portion of the page, featuring a large square and a smaller square within it.



La sezione aurea





La sezione aurea

Il valore della
sezione aurea è
circa 1.618

100 * 161.8

100 * 140

6

Esercizio



Esercizio

- Scrivere una funzione che accetta un parametro e restituisce il suo tipo
- Scrivere una funzione che accetta due parametri e ne restituisce il loro prodotto
- Scrivere una funzione che accetta due numeri e una stringa che identifica l'operazione da eseguire e ritorni il risultato corretto
- Scrivere una funzione che accetta una stringa e dice se è palindroma

7

jQuery

IL \$



jQuery ci mette a disposizione un alias per semplificare il suo utilizzo: \$.

\$ può essere usato come selettore degli elementi del DOM.

Funziona con gli stessi selettori di CSS e può usare anche le sue pseudoclassi.



Selezionare un elemento

```
<div id="mio-id">...</div>
<div class="mia-classe">...</div>

<script>
    $('#mio-id');
    $('.mia-classe');
</script>
```

\$ ci permette di selezionare uno o più elementi dal DOM e ottenere una referenza a loro semplicemente usando i classi selettori CSS.



Selettori CSS avanzati

div,p	Selects all <div> elements and all <p> elements
div > p	Selects all <p> elements where the parent is a <div> element
div + p	Selects all <p> elements that are placed immediately after <div> elements
p ~ ul	Selects every element that are preceded by a <p> element



Selettori CSS avanzati

input[name="nome1"]	Selects all <input> with attribute name equals to nome1
input:checked	Selects every checked <input> element
:not(p)	Selects every element that is not a <p> element
:hidden	Selects all elements which are hidden



Aggiungere / rimuovere classi

```
<div id="mio-id">...</div>
```

```
<script>  
    $('#mio-id').addClass('classe1');  
    $('#mio-id').removeClass('classe1');  
</script>
```

Per aggiungere una classe ad un elemento:

```
$('....').addClass('classe');
```

Mentre per rimuovere una classe da un elemento:

```
$('....').removeClass('classe');
```



Aggiungere / rimuovere classi

```
<div id="mio-id">...</div>
```

```
<script>  
    $('#mio-id').toggleClass('classe1');  
</script>
```

```
$(...).toggleClass('classe');
```

Aggiunge la classe 'classe' se non presente, mentre la toglie se già presente nell'elemento.



Aggiungere/rimuovere CSS inline

```
<div id="mio-id">...</div>
```

```
<script>
    $('#mio-id').css('background-color', 'red');
</script>
```

E' possibile aggiungere regole CSS inline semplicemente con:
\$('...').css('regola', 'valore');



Mostrare/nascondere un elemento

```
<div id="mio-id">...</div>
```

```
<script>
    $('#mio-id').hide();
    $('#mio-id').show();
</script>
```

Per nascondere un elemento, ovvero applicare `display: none;`
`$('#...').hide();`

Per mostrare un elemento, ovvero applicare `display: block;`
`$('#...').show();`



Testare la presenza di elementi

```
<div id="mio-id">...</div>

<script>
    if ($('#mio-id').length) {
        ...
    }
</script>
```

Per testare se uno o più elementi esistono all'interno della pagina:

`$('....').length`

Sarà maggiore di zero, ovvero uguale al numero di elementi selezionati



Attributi VS proprietà

Un attributo aggiunge delle informazioni ad un elemento (es. href).

Una proprietà descrive le caratteristiche di un elemento (es. checked)

```
<a href="#">link</a> <!-- href è un attributo -->
```

```
<input type="checkbox" checked> <!-- checked è una proprietà-->
```



Attributi

```
<a id="mio-a" href="https://google.com">...</a>
```

```
<script>
    console.log($('#mio-a').attr('href'));
</script>
```

Per accedere agli attributi:

```
$(...).attr('nome_attributo');
```

Per impostare un attributo

```
$(...).attr('nome_attributo', 'valore_attributo');
```

x Expression
not available

⚠ A cookie associated with this page was used to deliver cookies with cross-site requests and see more details at <https://google.com>

⚠ A cookie associated with this page was used to deliver cookies with cross-site requests and see more details at <https://google.com>

> |



Proprietà

```
<input type="checkbox" id="check">  
<script>  
    $('#check').prop('checked', true);  
</script>
```

Per accedere ottenere il valore di una proprietà:

```
$(...).prop('nome_ proprietà');
```

Per impostare un proprietà

```
$(...).prop('nome_proprietà', 'valore');
```



Attributi 'data-'

E' possibile usare degli attributi aggiuntivi non standard agli elementi (es. per contenere delle informazioni che ci fornisce il server) usando gli attributi data-.

```
<div data-mio-attributo="mio-valore"></div>
```



Attributi 'data-'

```
<div data-mio-attributo="mio-valore" id="mio-id"></div>  
<script>  
    console.log($('#mio-id').data('mio-attributo'));  
</script>
```

Per accedere ottenere il valore di un attributo data:

```
$(...).data('nome-senza-data');
```

Per impostare un attributo data:

```
$(...).data('nome-senza-data', 'valore');
```

The screenshot shows a browser's developer tools with the expression evaluator open. The expression being evaluated is '\$('#mio-id').data('mio-attributo')'. The result is displayed as 'not available' with a warning icon. Below the result, two log entries are shown, each with a warning icon and the text 'A cookie associated with this page was delivered via cookie delivery to another page, and see more details'. The most recent log entry shows the value 'mio-valore'.

x	Expression
not available	
⚠	A cookie associated with this page was delivered via cookie delivery to another page, and see more details
⚠	A cookie associated with this page was delivered via cookie delivery to another page, and see more details
mio-valore	



Modificare il contenuto

```
<div id="mio-id"><div>
<script>
  $('#mio-id').html($('<p>test</p>'));
</script>
```

test

Per accedere al contenuto:

```
$(...).html();
```

Per impostare il contenuto:

```
$(...).html('valore');
```



Creare un elemento

```
<script>  
    $('<p>Test</p>').appendTo($('body'));  
</script>
```

test

Per creare un elemento è sufficiente passare html come stringa a \$

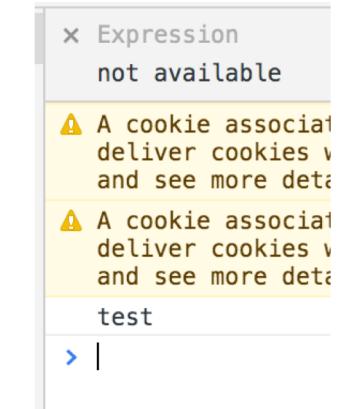
`$(...).appendTo(el)`

Permette di appendere un elemento ad un altro (oppure di muoverlo)



Accedere al contenuto di un input

```
<input id="mio-i" value="test">  
<script>  
    console.log($('#mio-i').val());  
</script>
```



Per accedere al contenuto:

```
$('#...').val();
```

Per impostare il contenuto:

```
$('#...').val('valore');
```



Eventi ed event handler

```
<button id="mio-id">Test</button>
<script>
    $('#mio-id').on('click', function () {
        alert('cliccato');
    });
</script>
```



Eventi ed event handler

jQuery permette di associare un event handler ad un evento, ed il click di un bottone. La sintassi base è:

```
$(..).on('evento', function () {  
    // Codice  
});
```

Allo stesso modo è possibile anche eliminare un event handler

```
$(..).off('evento');
```



Transazioni di jQuery

jQuery possiede alcune transazioni di base utili per mostrare o nascondere del contenuto:

- fadeIn /fadeOut
- slideDown/slideUp



\$(this)

Quando usate jQuery e siete all'interno di una funzione (ad esempio perchè state scrivendo un event handler) a volte capita di dover accedere all'elemento che ha generato l'evento.

jQuery mette a disposizione questa sintassi:
\$(this)



preventDefault();

Se provate a registrare un click event in un link il browser segue comunque il link dopo aver eseguito il vostro handler.

Per prevenire questo problema esiste una "funzione" capace di inibire il comportamento di default.

```
$('a').on('click', function (e) {  
    e.preventDefault();  
    // Codice  
});
```

8

Esercizio



Esercizio

- Creare una calcolatrice con tutti i tasti come numeri e che esegua le 4 operazioni fondamentali (+,-, *, /).
- La calcolatrice deve avere un tasto CE per cancellare l'operazione corrente e ricominciare da capo l'operazione.
- Il codice non deve avere il minor codice ripetuto (usate le funzioni).
- Stilate la calcolatrice con bootstrap/CSS

9

Esercizio



Esercizio

- Creare un form di registrazione con bootstrap. Il form deve avere nome, cognome, username, email, password e conferma password.
- Intercettare con jQuery l'invio del form (o del click sul pulsante di registrazione)
- Validare il form di registrazione in modo che il nome e il cognome siano lunghi almeno 3 caratteri, lo username abbia lunghezza compresa tra 7 e 20 caratteri, le due password coincidano e la mail sia valida.

Per validare la mail:

<https://stackoverflow.com/questions/46155/how-to-validate-an-email-address-in-javascript>

- Validate la password in modo che sia sicura (almeno un carattere maiuscolo, un carattere minuscolo, un numero e un carattere speciale)



Thanks!

Any questions?

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