

# *Javascript*

*Qual è il linguaggio migliore ?*

## *Caratteristiche distintive dei linguaggi:*

- *Basso Livello / Alto Livello*
- *Compilati / Interpretati*
- *Tipizzati / Non Tipizzati*
- *Funzionali / Ad Oggetti*
- *Bloccanti / Non bloccanti*

## *Caratteristiche distnitive di Javascript :*

- *Linguaggio di Programmazione di alto livello*
- *Interpretato (Dal Browser)*
- *Client-Side*
- *Non tipizzato (tipizzazione debole e dinamica)*
- *Funzionale + Oggetti*
- *Asincrono/Non Bloccante*

# Javascript – Introduzione

*Javascript*

# Come inserire javascript in un file html

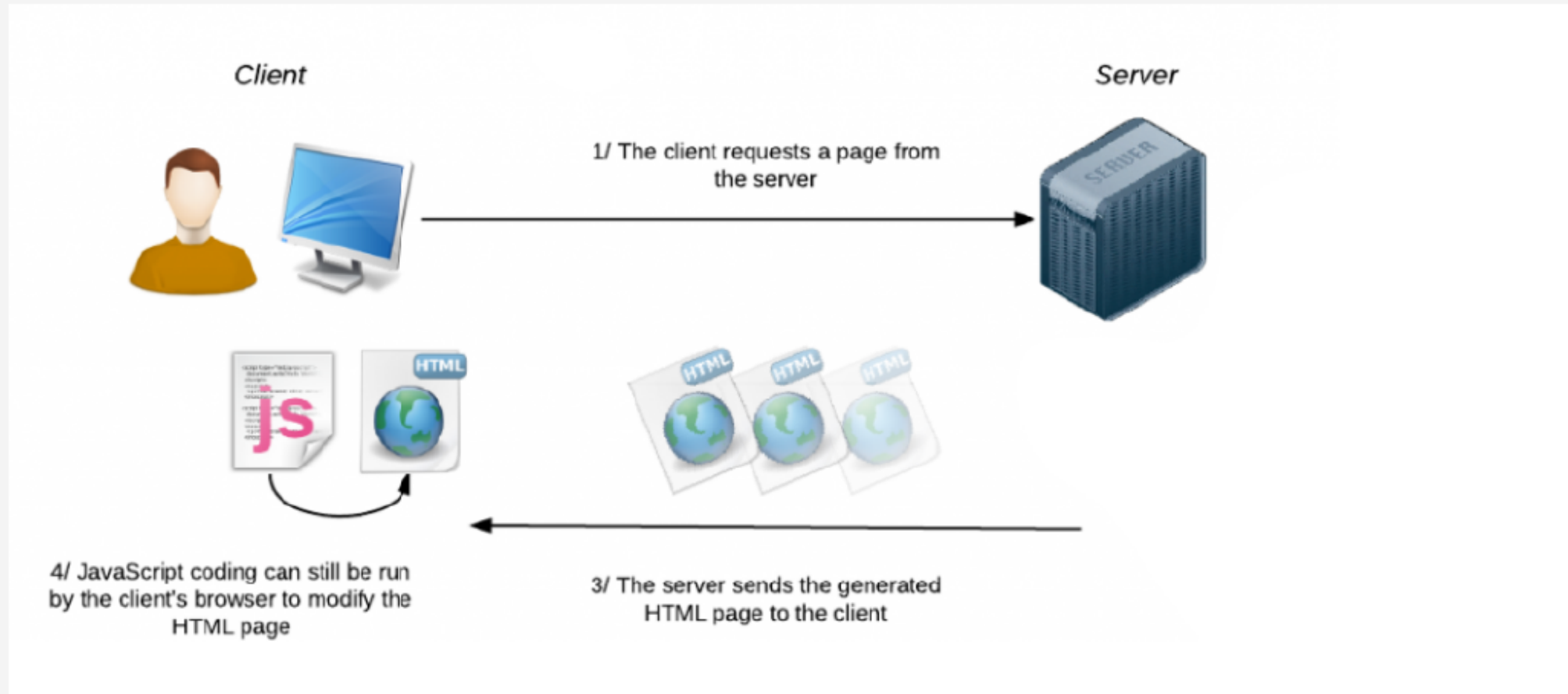
Direttamente in un tag script:

```
6 <script>
7     console.log('I\'m in a script tag')
8 </script>
```

In un file separato




























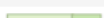

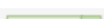

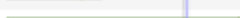
```
3 <head>
4     <script src="js/ajax.js"></script>
5 </head>
```

*Il codice viene interpretato da cima a fondo. Il caricamento della pagina viene interrotto fino a che tutto il codice non viene valutato.*



*Il file html è il primo oggetto ricevuto dal server*

# Client Side - Asincronia

Name	Stat...	Type	Size	Time	Waterfall	2.00 s	4.00 s	6.00 s	8.00 s	10.00 s
 ac-globalnav.built.js	200	script	33.1 KB	795 ms						
 ac-analytics.js	200	script	58.9 KB	945 ms						
 ac-localnav.built.js	200	script	10.8 KB	604 ms						
 ac-globalfooter.built.js	200	script	3.6 KB	572 ms						
 localeswitcher.built.js	200	script	9.3 KB	610 ms						
 main.built.js	200	script	139 KB	1.69 s						
 modal.css	200	stylesheet	11.3 KB	569 ms						
 autofilms.built.js	200	script	69.7 KB	1.05 s						
 sf-pro-text_regular.w...	200	font	62.6 KB	3.21 s						
 image_large.svg	200	svg+xml	1.1 KB	1.65 s						
 image_large.svg	200	svg+xml	1.5 KB	1.40 s						
 image_large.svg	200	svg+xml	1.2 KB	321 ms						
 image_large.svg	200	svg+xml	1.2 KB	445 ms						
 image_large.svg	200	svg+xml	1.2 KB	708 ms						
 image_large.svg	200	svg+xml	1.1 KB	711 ms						
 image_large.svg	200	svg+xml	1.1 KB	1.77 s						

*Ogni link esterno (immagini, script) genera una ulteriore richiesta http*



Linguaggio debolmente tipizzato (loosely typed):

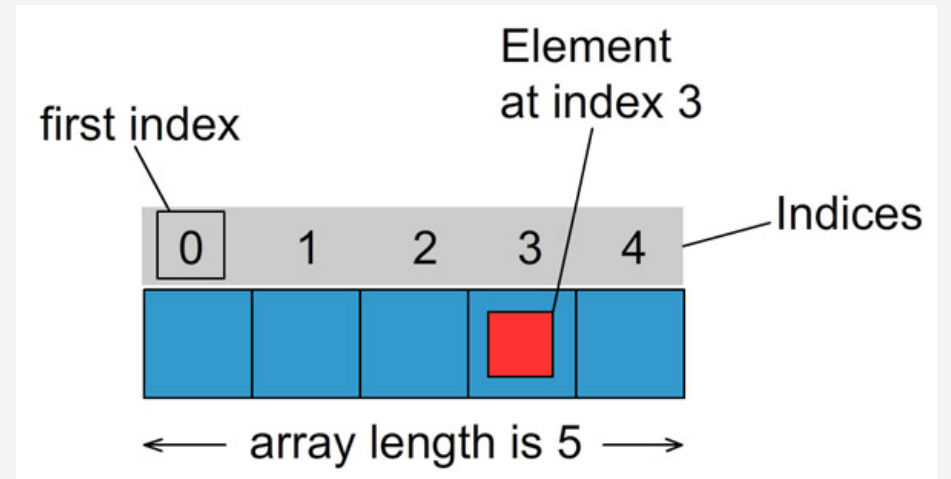
- No definizione **Tipo** in fase di dichiarazione
- Possibile riassegnare la stessa variabile cambiando **Tipo**

```
13 // Dichiarazione
14 var nome;
15 nome = 'name'
16 nome = 'my_' + nome
17
18 // Dichiarazione + assegnazione
19 var nome = 'my_name'
20 // Riassegnazione con diverso tipo
21 nome = 2
```

# Array

Collezione di variabili identificate da una posizione (Indice numerico )

Il primo indice è 0



```
63 // Costruzione
64 var lista = new Array()
65 lista[1] = 'a'
66
67 // Costruzione + assegnazione
68 var lista = ['a', 'b', 'c', 'd']
69 lista[1] // 'b'
70 lista[1] = 'z' // ['a', 'z', 'c', 'd']
71 lista[4] = 'x' // ['a', 'z', 'c', 'd', 'x']
```

- Riutilizzare lo stesso codice più volte
- Organizzare/Strutturare il codice
- Eseguire operazioni asincrone

```
87 // Definizione
88 ✓ function stampa_doppio(stampami, fine){
89     console.log( stampami + ' ' + stampami + fine)
90 }
91 // Esecuzione
92 stampa_doppio('ciao', '!')           // "ciao ciao!"
93 stampa_doppio('bye', '...')         // "bye bye..."
```

# Funzioni – valore di ritorno

La funzione può ritornare un valore usando il **return** statement

```
96 // Definizione
97 function doppio(stampami, fine){
98     return stampami + ' ' + stampami + fine
99 }
100 // Esecuzione
101 var risultato = doppio('ciao', '!') + doppio('bye', '...')
102 console.log(risultato) // "ciao ciao!bye bye..."
```

Funzioni senza **return** ritornano **undefined**

# Funzioni – valore di ritorno

La funzione può ritornare un valore usando il **return** statement

```
96 // Definizione
97 function doppio(stampami, fine){
98     return stampami + ' ' + stampami + fine
99 }
100 // Esecuzione
101 var risultato = doppio('ciao', '!') + doppio('bye', '...')
102 console.log(risultato) // "ciao ciao!bye bye..."
```

Funzioni senza **return** ritornano **undefined**

Funzione passata come parametro ad un'altra funzione

```
67 function logCiao(){
68     console.log('Ciao!!!!')
69 }
70 setTimeout(logCiao , 5000);           // ...5sec -> 'Ciao!!!'
71
72 // Utilizzando una funzione anonima
73 setTimeout( function(){ console.log('I am anonymus') }, 3000 );
74                                     // ...3sec -> 'I am anonymus'
```

*Asincronia : Leggero in console prima “I am anonymus” e dopo “Ciao”*

Funzione passata come parametro ad un'altra funzione

```
72     function add_title(title, sonoUnaFunzione){  
73         return title + ' ' + sonoUnaFunzione()  
74     }  
75  
76     function ritornaMario(){  
77         return 'Mario Rossi '  
78     }  
79     console.log(add_title('Mr.', ritornaMario)); // "Mr. Mario Rossi"
```

- Contenitore di variabili e funzioni
- Raggruppare logicamente variabili e funzioni

```
123 // Definizione
124 var persona = {
125     // proprietà
126     nome : 'tizio',
127     cognome : 'caio',
128     anni : 20,
129     // metodi
130     stampaNomeCompleto : function(){
131         console.log(persona.nome + ' ' + persona.cognome)
132     }
133     getNomeCompleto : function(){ return persona.nome+' '+persona.cognome }
134 }
```



```
135 // Lettura alle proprietà
136 console.log(persona.nome) // "tizio"
137 // Modifica proprietà
138 persona.nome = "sempronio"
139 console.log(persona.nome) // "sempronio"
140
141 // Esecuzione metodo
142 persona.stampaNomeCompleto() // "tizio sempronio"
```

*Variabili prendono il nome di **Proprietà***

*Funzioni prendono il nome di **Metodi***

# Javascript – Introduzione

*Tipi*

Linguaggio debolmente tipizzato: esistono dei **Tipi** di dato

```
3  typeof "John"           // "string"
4  typeof 3.14              // "number"
5  typeof myCar             // "undefined" (non assegnata)
6  typeof false            // "boolean"
7  typeof NaN              // "number"
8  typeof [1,2,3,4]         // "object"
9  typeof {nome:'tizio',eta:34} // "object"
10 typeof new Date()        // "object"
11 typeof function () {}   // "function"
12 typeof null              // "object"
```

## Conversione implicita

```
25 // Implicita
26 var nome = 'tizio_' + 2 // "tizio_2" (string)
27 var eta = '30' + 2      // "302" (string)
28 var numero = 30 + true  // 31 (number)
```

*Priorità : string > number > boolean | null | undefined | false*

## Conversione esplicita

```
29 // Esplicita
30 var eta = parseInt('30') + 2 // 32 (number)
```

# Javascript – Introduzione

## *Operatori*

Category	Operator	Name/Description	Example	Result
Arithmetic	+	Addition	3+2	5
	-	Subtraction	3-2	1
	*	Multiplication	3*2	6
	/	Division	10/5	2
	%	Modulus	10%5	0
	++	Increment and then return value	X=3; ++X	4
		Return value and then increment	X=3; X++	3
	--	Decrement and then return value	X=3; --X	2
		Return value and then decrement	X=3; X--	3
Logical	&&	Logical “and” evaluates to true when both operands are true	3>2 && 5>3	False
		Logical “or” evaluates to true when either operand is true	3>1    2>5	True
	!	Logical “not” evaluates to true if the operand is false	3!=2	True
Comparison	==	Equal	5==9	False
	!=	Not equal	6!=4	True
	<	Less than	3<2	False
	<=	Less than or equal	5<=2	False
	>	Greater than	4>3	True
	>=	Greater than or equal	4>=4	True
String	+	Concatenation(join two strings together)	“A”+”BC”	ABC

# Operatori - Uguaglianza

	true	false	1	0	-1	"true"	"false"	"1"	"0"	"-1"	""	null	undefined	Infinity	-Infinity	[]	{}	[[]]	[0]	[1]	NaN
true																					
false																					
1																					
0																					
-1																					
"true"																					
"false"																					
"1"																					
"0"																					
"-1"																					
""																					
null																					
undefined																					
Infinity																					
-Infinity																					
[]																					
{}																					
[[]]																					
[0]																					
[1]																					
NaN																					

## Esempi (==)

- *false == 0* → true
- *false == "0"* → true
- *false == ""* → true
- *false == false* → true

## Esempi (===)

- *false === 0* → false
- *false === "0"* → false
- *false === ""* → false
- *false === false* → true

# Javascript – Introduzione

*Controllo*



## Costrutto if

```
42  if( a == b ){  
43      console.log('a è uguale b')  
44  }  
45  else if (a == c){  
46      console.log('a è uguale c')  
47  }  
48  else {  
49      console.log('a non è uguale')  
50  }
```

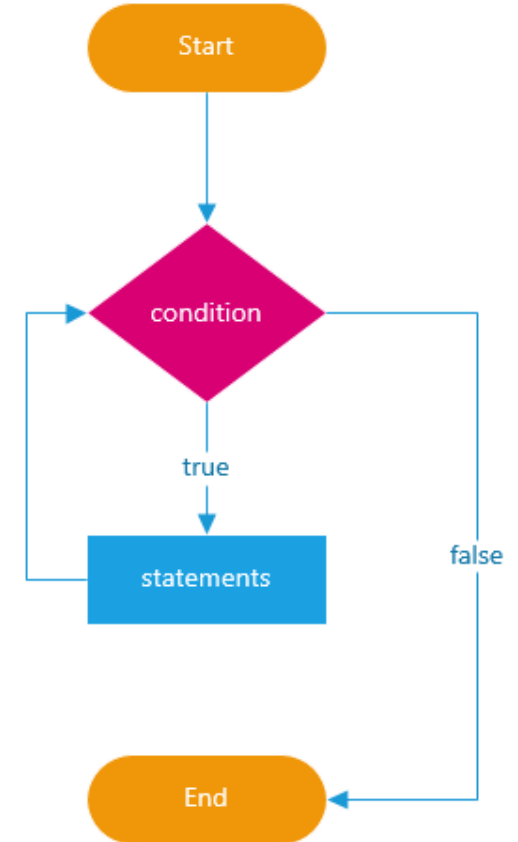
## Operazioni booleane

```
53  true && false           // false
54  true || false           // true
55  "0" || 1                 // true
56  "nome" && 0               // false
57  "nome" == "nome" && !(0 === "0") // false
58  "nome" == "nome" && 0 === "0" || 0 < 1 // true
59  "nome"                   // true
60  !"nome"                  // false
```

# Ciclo while

- Statement eseguito per un numero di volte non definito
- Si ferma solo quando la condizione è falsa
- Pericolo di eseguire un “Loop infinito”

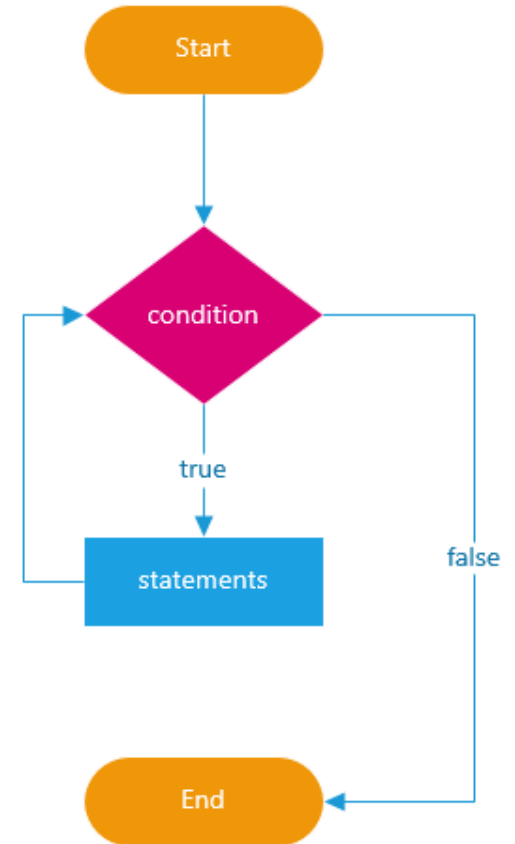
```
74  var x = 0;  
75  while( x < 10 /* condizione */)   
76  { /* statement */  
77      x = x+2  
78      console.log(x)  
79  }
```



## Ciclo for

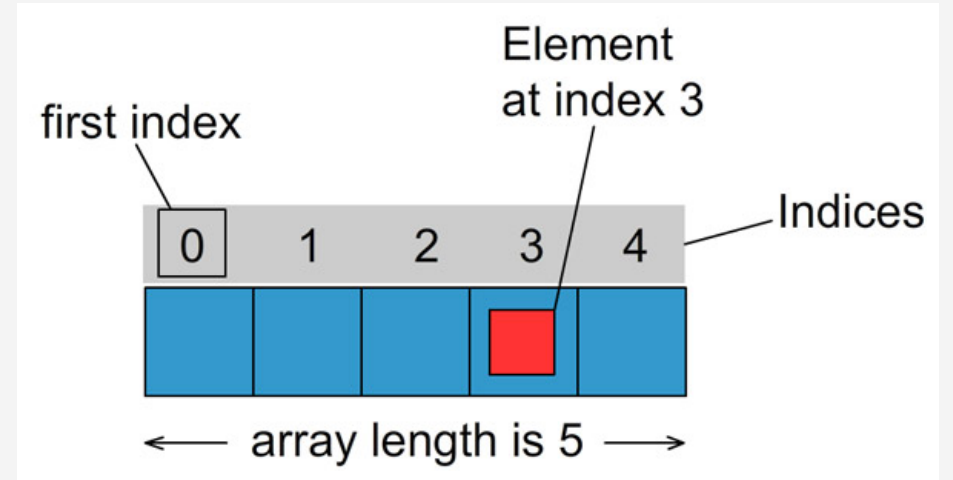
- Equivalente al ciclo while
- Si ferma solo quando la condizione è falsa
- Pericolo minore di eseguire un “Loop infinito”

```
81  for(var x = 0; x < 10; x=x+2){  
82      console.log(x)  
83  }
```



## Ciclo “for in”

Itera gli indici relativi agli elementi di un array



```
86  var lista = ['a', 'b', 'c', 'd']
87  for(index in lista){
88      console.log( lista[index] )
89  }
```

## Ciclo “for in”

E in grado di iterare anche le chiavi di un oggetto

```
81  var persona = { nome: 'tizio', cognome: 'caio', anni: 20 }  
82  for(index in persona){  
83      console.log( persona[index] )  
84  }
```

*Il ciclo “for in” itera anche le chiavi associate ai **metodi***

## *Oggetti*

Le stringhe sono oggetti con specifiche proprietà e metodi

```
149 // proprieta
150 var nome = 'tizio'
151 console.log( nome.length )           // 5
152
153 // metodi
154 var nomeMaiuscolo = nome.toUpperCase()
155 console.log( nomeMaiuscolo )         // "TIZIO"
156 console.log( nome )                  // "tizio"
```

*I metodi non modificano la stringa originale ma ne ritornano una nuova*



## METODI DELLE STRINGHE

`charAt()` Returns the character at the specified `index` (position)

`endsWith()` Checks whether a string ends `with` specified string/characters

`includes()` Checks whether a string contains the specified string/characters

`indexOf()` Returns the position `of` the first found occurrence

`lastIndexOf()` Returns the position `of` the last found occurrence

`repeat()` Returns a `new string with` a specified number `of` copies `of` string

`replace()` Searches a string `for` a specified value, or a regular expression, and returns a `new string` where the specified values are replaced

`search()` Searches a string `for` a specified value, or regular expression, and returns the position `of` the match

`slice()` Extracts a part `of` a string and returns a `new string`

`split()` Splits a string into an array `of` substrings

`startsWith()` Checks whether a string begins `with` specified characters

`substring()` Extracts the characters, between two specified indices

`toLowerCase()` Converts a string to lowercase letters

`toUpperCase()` Converts a string to uppercase letters

`trim()` Removes whitespace from both ends `of` a string

Gli array sono oggetti con specifiche proprietà e metodi

```
199 // proprietà
200 var lista = ['a', 'b', 'c', 'd']
201 console.log( lista.length )           // 4
202
203 // metodi
204 lista.unshift('z')
205 console.log( lista )                  // ['z', 'a', 'b', 'c', 'd']
206 console.log( lista.indexOf("a") )     // 1
```

*Alcuni metodi modificano l'array originale*

## ARRAY METHODS

`concat()` Joins two or more arrays, and returns a copy of the joined arrays  
`fill()` Fill the elements in an array with a static value  
`findIndex()` Returns the index of the first element in an array that pass a test  
`indexOf()` Search the array for an element and returns its position  
`join()` Joins all elements of an array into a string  
`pop()` Removes the last element of an array, and returns that element  
`push()` Adds new elements to the end of an array, and returns the new length  
`shift()` Removes the first element of an array, and returns that element  
`slice()` Selects a part of an array, and returns the new array  
`sort()` Sorts the elements of an array  
`splice()` Adds/Removes elements from an array  
`unshift()` Adds new elements to the beginning of an array,  
and returns the new length

È possibile creare oggetti a partire da un modello comune (classe)

```
224 // Definizione
225 function Persona(nome, cognome){
226     this.nome = nome;
227     this.cognome = cognome
228     this.stampa = function(){
229         console.log( this.nome + ' ' + this.cognome )
230     }
231 }
232 // Costruzione
233 var ironman = new Persona('Tony', 'Stark')
234 var spiderman = new Persona('Peter', 'Parker')
235
236 ironman.stampa() // 'Tony Stark'
237 spiderman.stampa() // 'Peter Parker'
```

# Oggetti - Prototype

È possibile aggiungere metodi alla classe usando la proprietà **prototype**

```
240 // [...]
241 // Aggiungo tramite prototype
242 persona.prototype.upperName = function(){
243     this.nome.toUpperCase() + this.cogome.toUpperCase()
244 }
245
246 console.log(ironman.upperName()) // "TONY STARK"
247 console.log(spiderman.upperName()) // "PETER PARKER"
```

*Il nuovo metodo è aggiunto automaticamente a tutti gli oggetti costruiti a partire dalla classe*