

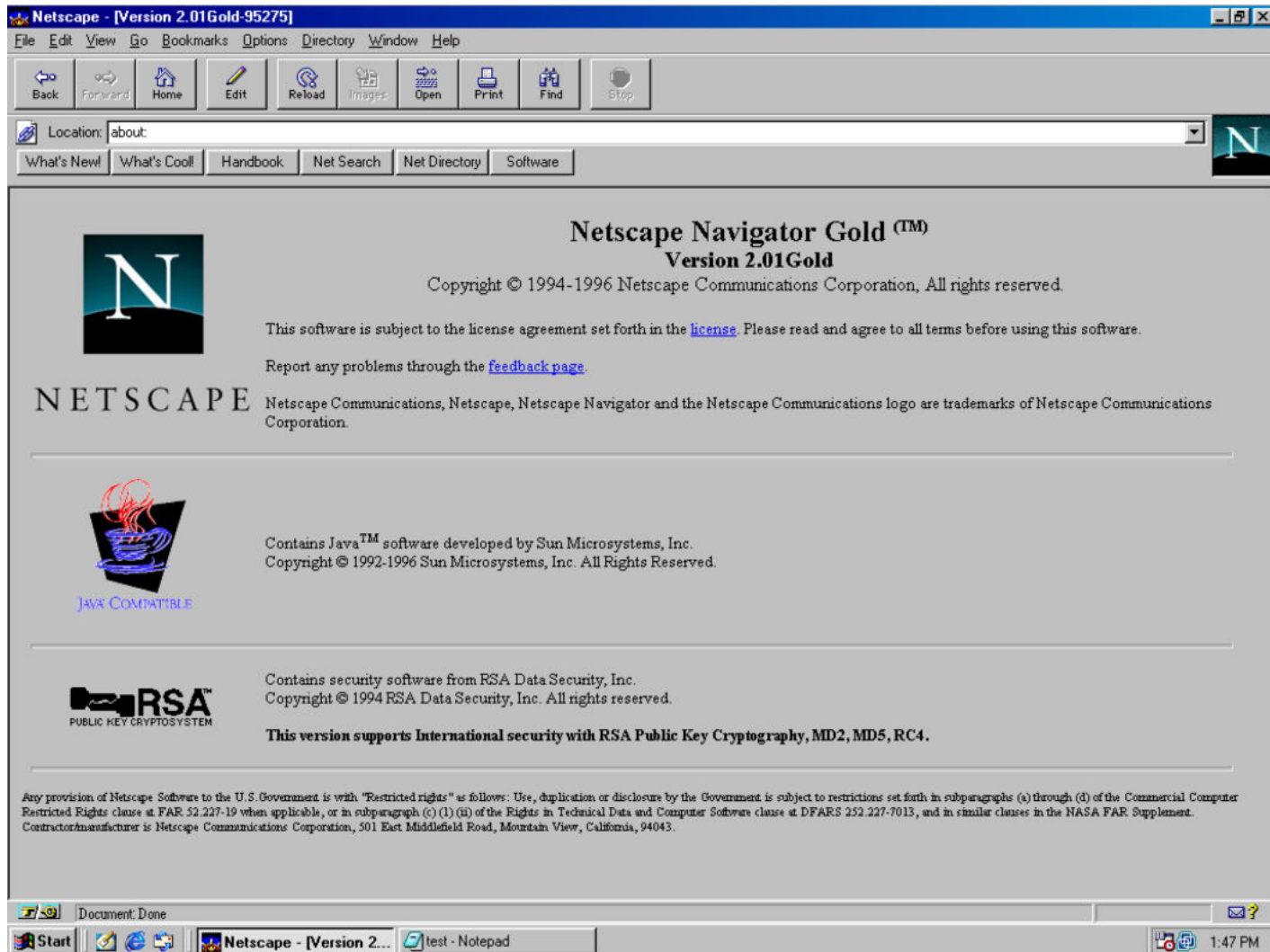


JavaScript

Lorenzo Bracciale

Cos'è Javascript?

- E' un linguaggio di programmazione creato nel 1995 da Brendan Eich (Netscape)
 - Originariamente chiamato "Livescript" / Mocha: nato come linguaggio "semplice" per i non sviluppatori (designers/scripters/amateurs)
 - Fatto in 10 giorni (!) - "make web pages alive"
- Javascript != Java (chiamato così solo per marketing!)
- Standard nel 1996 da European Computer Manufacturer's Association (ECMA)
 - Chiamato ECMAScript - ISO/IEC 16262
- Versione attuale dello standard: ECMAScript 2020



Oggi

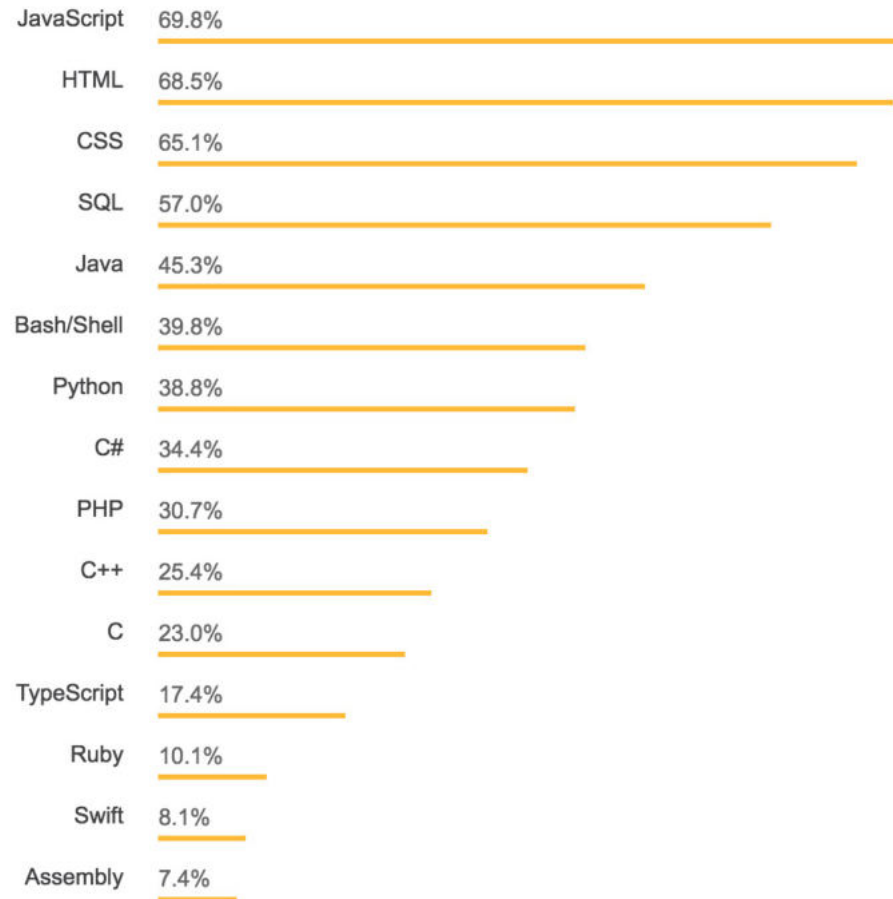


Most Popular Technologies

Programming, Scripting, and Markup Languages

All Respondents

Professional Developers

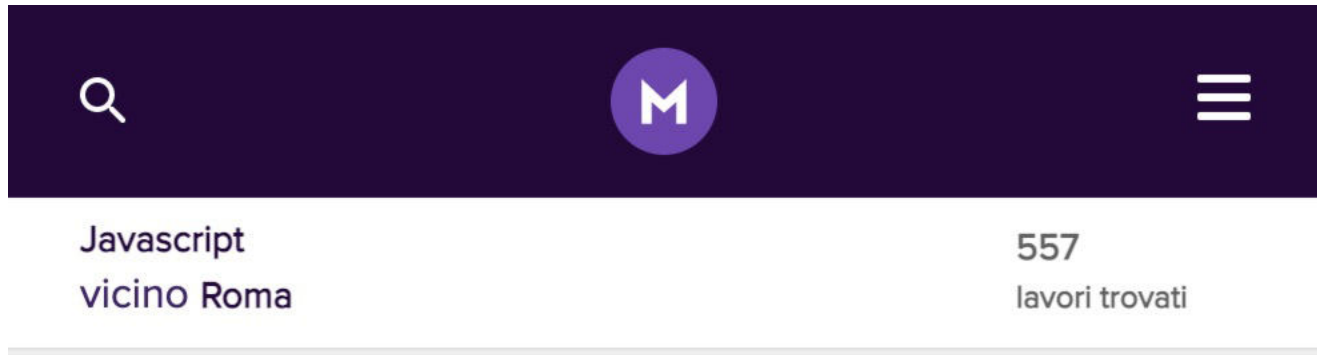


Stackoverflow survey

Javascript is the most commonly used programming language since 8 years

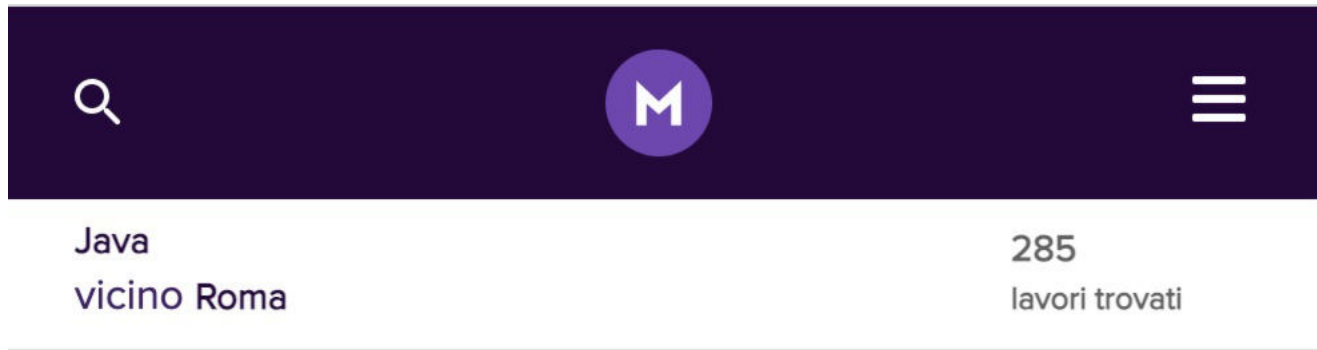
Una competenza richiesta

Lavoro "javascript" a Roma: *monster.it*



Javascript
vicino Roma

557
lavori trovati



Java
vicino Roma

285
lavori trovati

Dove studiare

- Libro di testo
- Documentazione online
 - <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
 - <https://javascript.info/>
- Libri specifici



Ruolo di Javascript

- Client side
 - praticamente monopolista!
- Server side
 - NodeJs

Struttura (HTML)

Presentazione (CSS)

Comportamento (JS)



Cosa fa javascript: instant search

Web [Images](#) [Videos](#) [Maps](#) [News](#) [Shopping](#) [Gmail](#) [more](#) ▼



 Everything
  Images
  Videos
 ☐ More

☐ Show search tools

weather
 weather
 walmart
 white pages
 wikipedia
 W

About 1,110,000,000 results (0.23 seconds)

Weather for New York, NY - [Change location](#) - [Add to iGoogle](#)

89°F | °C
 Current: **Partly Cloudy**
 Wind: W at 16 mph
 Humidity: 21%

Wed	Thu	Fri	Sat
			
82°F 60°F	76°F 58°F	73°F 58°F	78°F 63°F

Detailed forecast: [The Weather Channel](#) - [Weather Underground](#) - [AccuWeather](#)

[National and Local Weather Forecast, Hurricane, Radar and Report](#) ☆

The **Weather Channel** and **weather.com** provide a national and local **weather** forecast for cities, as well as weather radar, street and business closures.

Cosa fa javascript: chat

Nuovo messaggio

A: Ingegneria di Internet x

Scrivi un messaggio come Lorenzo Bracciale...

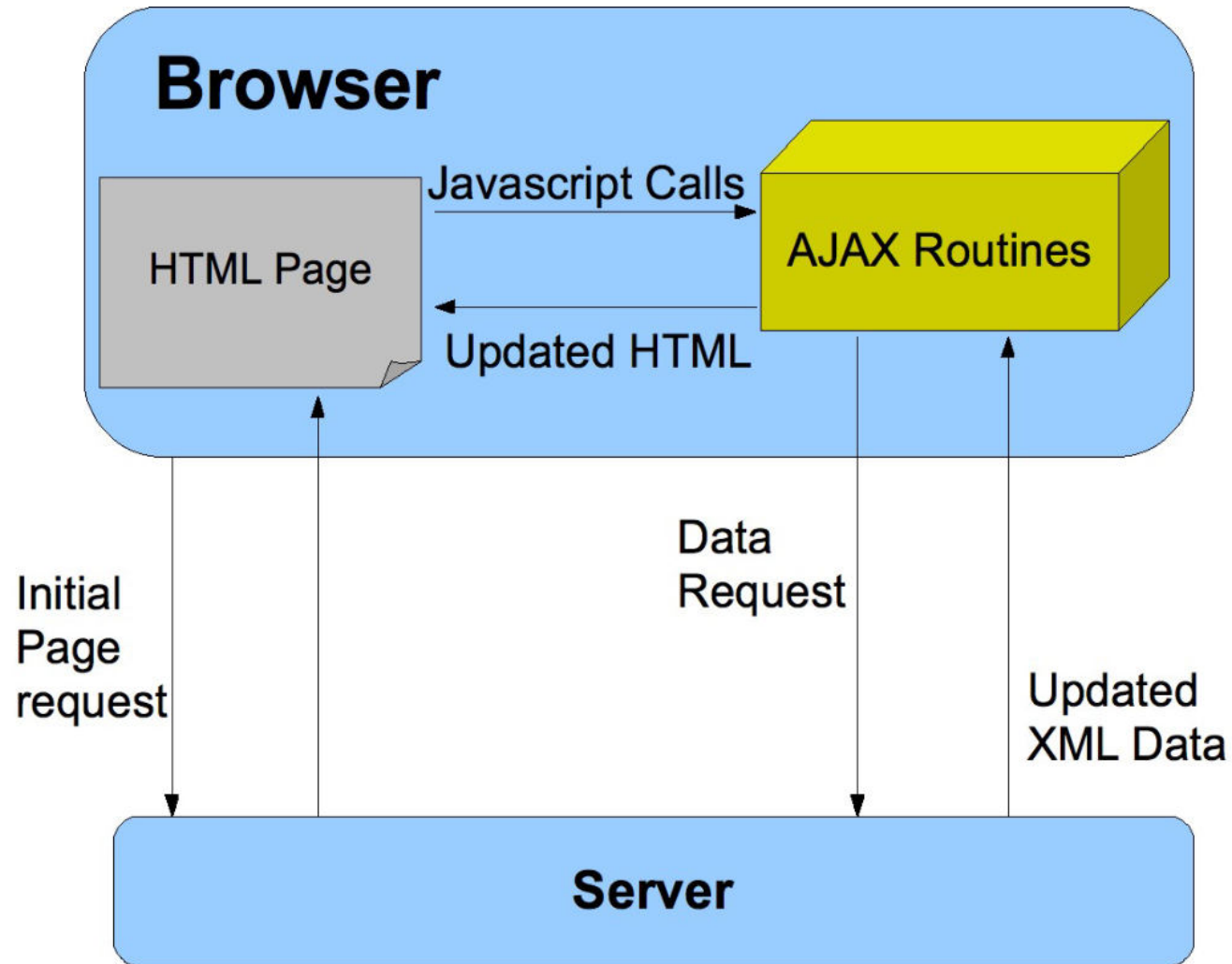
 Aggiungi file

 Aggiungi foto

Invia

Cosa fa javascript: **AJAX**

(Asynchronous JavaScript and XML,)



Cosa fa javascript

- Cambiare elementi e stili in una pagina
 - Ad es: aggiungere classi css in risposta a eventi generati dall'utente (click, scroll ecc) → DOM manipulation (more later)
- Comunicazione asincrona
 - Invio dati senza ricaricare la pagina o senza interazione con l'utente
 - Il paradigma è spesso chiamato AJAX
- Altro
 - Test browser capabilities ed adattamento (polyfills)
 - Concorrente APP mobile?
 - Cordova, Phonegap?

Cosa fa (e non fa) Javascript

Cosa fa

- Modifica elementi della pagina
- Interagisce con un server remoto
- Reagisce ad azioni dell'utente
- Imposta cookie e contenuti locali (file storage)
- Disegna sulla pagina

Cosa non fa (nei browser)

- Accede ai file locali del computer
- Interagisce con qualunque server remoto

JS VM

Javascript lato server: NodeJS

- Nel 2008 "Chromium Project" ha creato un motore javascript open source chiamato V8
- Nel 2009, V8 è stato usato per creare un **back-end** JavaScript runtime environment chiamato **Node.js**
- Ha permesso di usare uno stesso linguaggio sia client-side che server-side

Javascript Frameworks (client side)



- ExtJS (2009)
- Knockout (2012)
- Backbone (2013)
- Angular (2014)
- React (2015)
- Vue.js (2017)

Studieremo' "vanilla javascript": perché?

Download

Ready to try *Vanilla JS*? Choose exactly what you need!

<input checked="" type="checkbox"/> Core Functionality	<input type="checkbox"/> DOM (Traversal / Selectors)
<input type="checkbox"/> Prototype-based Object System	<input type="checkbox"/> AJAX
<input type="checkbox"/> Animations	<input type="checkbox"/> Event System
<input type="checkbox"/> Regular Expressions	<input type="checkbox"/> Functions as first-class objects
<input type="checkbox"/> Closures	<input type="checkbox"/> Math Library
<input type="checkbox"/> Array Library	<input type="checkbox"/> String Library

Options

<input type="checkbox"/> Minify Source Code	<input type="checkbox"/> Produce UTF8 Output
<input type="checkbox"/> Use "CRLF" line breaks (Windows)	

Final size: 0 bytes uncompressed, 25 bytes gzipped. ☐ Show human-readable sizes

[Download](#)

Testimonials

"Vanilla JS is the lowest-overhead, most comprehensive framework I've ever used."

Speed Comparison

Here are a few examples of just how fast *Vanilla JS* really is:

Retrieve DOM element by ID

	Code	ops / sec
<i>Vanilla JS</i>	<code>document.getElementById('test-table');</code>	12,137,211
Dojo	<code>dojo.byId('test-table');</code>	5,443,343
Prototype JS	<code>\$('test-table')</code>	2,940,734
Ext JS	<code>delete Ext.elCache['test-table']; Ext.get('test-table');</code>	997,562
jQuery	<code>\$jq('#test-table');</code>	350,557
YUI	<code>YAHOO.util.Dom.get('test-table');</code>	326,534
MooTools	<code>document.id('test-table');</code>	78,802

Retrieve DOM elements by tag name

	Code	ops / sec
<i>Vanilla JS</i>	<code>document.getElementsByTagName("span");</code>	8,280,893
Prototype JS	<code>Prototype.Selector.select('span', document);</code>	62,872
YUI	<code>YAHOO.util.Dom.getElementsByTagName(function(){return true;},'span');</code>	48,545
Ext JS	<code>Ext.query('span');</code>	46,915
jQuery	<code>\$jq('span');</code>	19,449
Dojo	<code>dojo.query('span');</code>	10,335
MooTools	<code>Slick.search(document, 'span', new Elements);</code>	5,457

Compatibilità

Can I use [? ⚙ Settings](#)

✕	Feature: ECMAScript 2015 (ES6)
---	--------------------------------

ECMAScript 2015 (ES6) - OTHER

Usage

% of all users

Global

$$94.41\% + 2.91\% = 97.32\%$$

Support for the ECMAScript 2015 specification. Features include Promises, Modules, Classes, Template Literals, Arrow Functions, Let and Const, Default Parameters, Generators, Destructuring Assignment, Rest & Spread, Map/Set & WeakMap/WeakSet and many more.

Current aligned

Usage relative

Date relative

Filtered

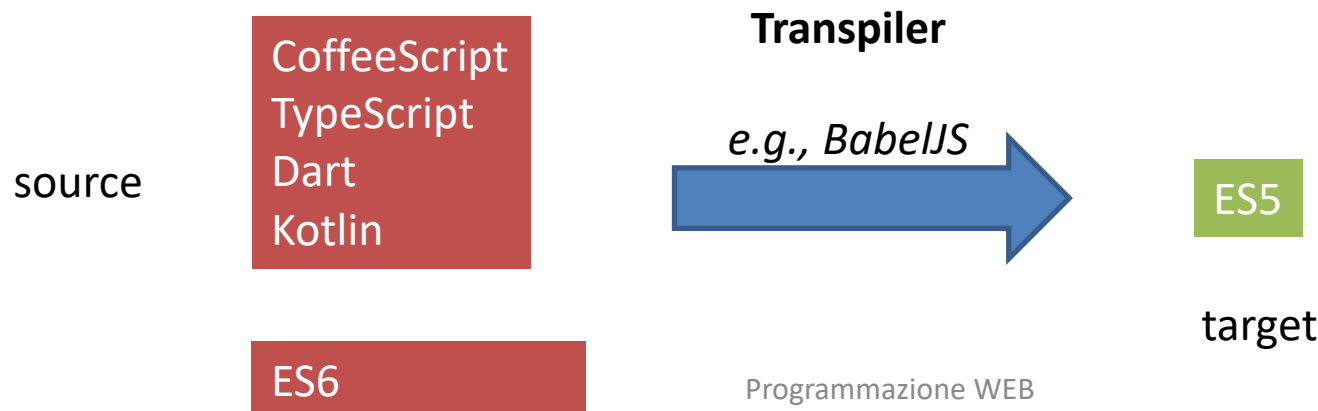
All

[illegible]

<https://caniuse.com/>

Transpilers

- Servono a tradurre linguaggi differenti, in una specifica versione di JS (source-to-source translators)
- Perché?
 - Permettono la scrittura dei programmi in diversi modi, più comodi per alcuni utenti
 - Backward compatibility (ad es. ES6 -> ES5)



Backward compatibility: polyfill

- I **polyfill** sono librerie js che adattano del codice (html, css, js) per renderlo retrocompatibile con standard "vecchi".

Esempi

HTML5 polyfill: rende una pagina HTML5 compatibile con vecchi browser non HTML5

MODERNIZR: testa la presenza di features nel browser e carica polyfill se servono

SELECTIVIZR: fa capire a vecchie versioni di IE query CSS3 complesse

Caratteristiche del linguaggio

- **Dynamic**: non è compilato, gira in una “macchina virtuale”
- **Loosely typed**: non bisogna dire che tipo ha una variabile
- **Case-sensitive**: aTTenZione allE MaiUscole!

Javascript garbage collector

- Un “garbage collector” rimuove le variabili che non ci servono dalla memoria automaticamente
 - Lo capisce quando non abbiamo più “riferimenti” ad un oggetto
 - Possiamo dichiarare nuove variabili dinamicamente senza preoccuparci* di rimuoverle dalla memoria

* in realtà un po' dovremmo preoccuparci, anche qui possiamo fare dei “memory leak”...
more info: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management

Come è nato javascript?

- <https://www.youtube.com/watch?v=Sh6IK57Cuk4>



Hello world!

Eseguiamo del codice javascript

- Interactive shell
 - Client side: browser console
 - Server side: node console
- Esecuzione di file ".js"
 - Client side: `<script>` tag
 - Server side: launch with node

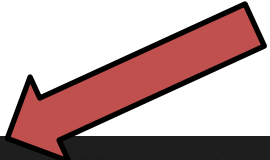
Client side: browser console

The screenshot shows a web browser window. The page title is "Home Page" and the main content is "Corso di Programmazione Web". Below this, there is a navigation bar with "P" and "PROGRAMMAZIONE WEB". The browser's developer tools are open, showing the "Console" tab. A red arrow points to the console log, which contains the following messages:

```
console.log("ciao a tutti gli studenti di programmazione web");  
ciao a tutti gli studenti di programmazione web  
undefined
```

The browser's address bar shows the URL "http://localhost:3000/". The page also features a search bar, a "LINK UTILI" section with links to "Materiale Didattico", "Iscriversi alla mailing list", and "Gruppo Telegram", and a footer with "Programma di Programmazione Web".

Server side: node console



```
aquilanteII:Downloads lorenzo$ node  
Debugger listening on ws://127.0.0.1:56218/75e7d2c9-84fe-4eb2-8be7-ef8789218c7e  
For help, see: https://nodejs.org/en/docs/inspector  
Debugger attached.  
> console.log("ciao");  
ciao  
undefined  
> █
```

Aggiungere Javascript a una pagina web

```
<script>
```

```
// Scrivi qui il tuo codice javascript
```

```
</script>
```

embedded

external

```
<script src="my_script.js">
```

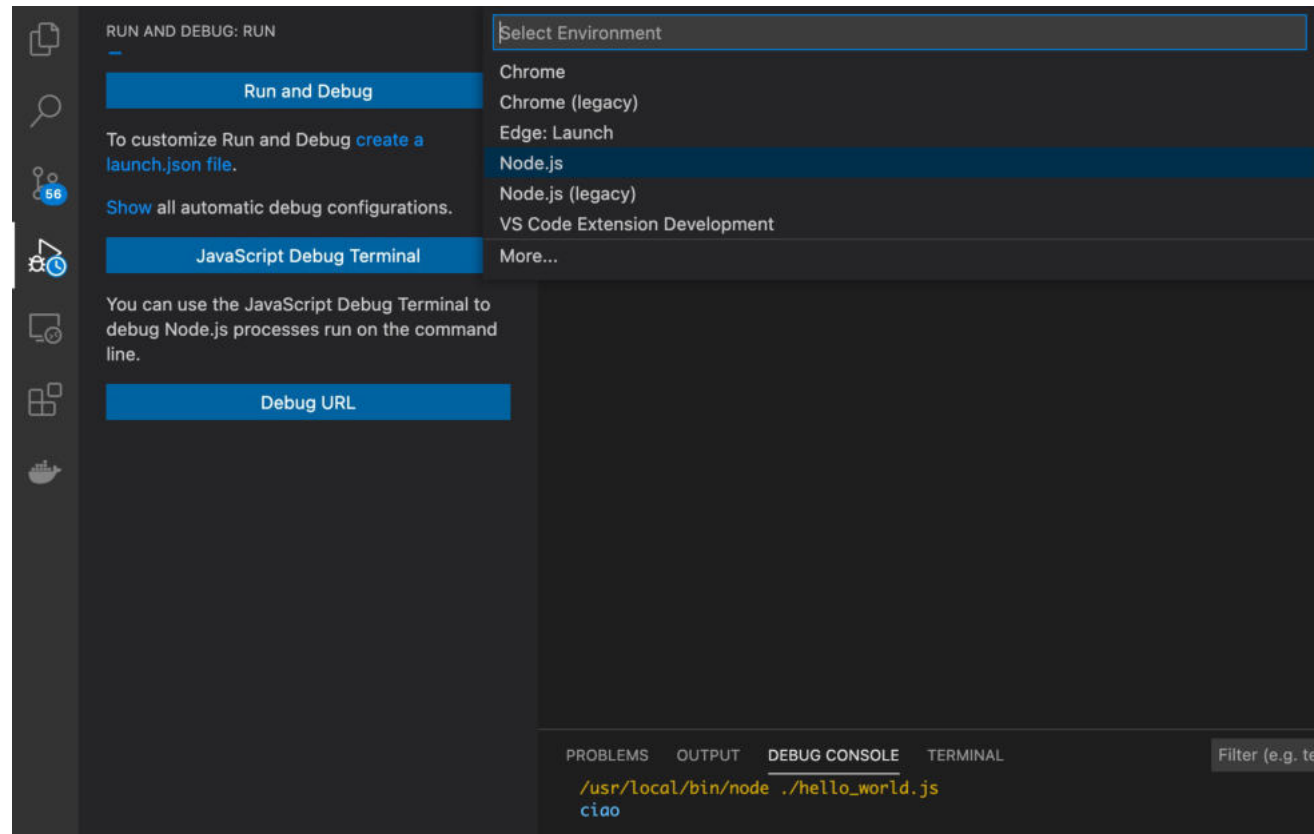
```
</script>
```

Solitamente inclusi in <head> o alla fine del <body>

Eseguire js code in Visual Studio (node)

- 1) Write a .js file
- 2) Run and Debug with Node.js

```
JS hello_world.js ×
JS hello_world.js
1 console.log("ciao");
2
```



Lavorare con le variabili

Variabili

Semicolon (";") alla fine di ogni statement è opzionale (Automatic Semicolon Insertion - ASI - su [ECMA Specification](#))



// questo è un commento

let myName = "lorenzo"; // variabile con stringa

let myNumber = 5; // variabile con intero

let myVar; // variabile non inizializzata

myVar = myNumber; // assegnazione

console.log(myVar); //stampiamo il risultato

Possiamo dichiarare variabili con *var* o *let*.
Useremo spesso *let*, più avanti spiegheremo la differenza.

Tipi di dato

Per vedere il tipo: `typeof(test)`

Ogni dato appartiene ad un tipo:

- ma non serve specificarlo nella dichiarazione (*dynamically typed*)
- Ci sono tipo "primitivi" (questi) e tipi "complessi" (oggetti, array, funzioni)

<code>let test = 5;</code>	la variabile è un numero . (anche "float": 5.123) Operazioni: + - * / % ** (es 5 + 3.1)
<code>let test = "ciao";</code>	la variabile è una stringa . (singoli o doppi apici) Concatenazione con + (es: "ciao" + ' a tutti')
<code>let test = true;</code>	variabile booleana Inverso con ! (es !test è false)
<code>let test;</code>	la variabile è undefined
<code>let test = null;</code>	la variabile è null

Metodi

- Una variabile ha dei metodi, a seconda del tipo

dichiarazione	tipo	esempio di metodi
let v = "ciao";	stringa	v.toUpperCase()
let v = 3.14	floating point number	v.toFixed()

Potremmo quindi scrivere anche:
`"ciao".toUpperCase()`

Operazioni matematiche di base

```
let i = 0, j = 2;
```

```
i = 2 + j * 4; // i = 10
```

```
i++; // i = 11
```

// Utilizzo delle parantesi per precedenza

```
i = (2 + j) * 4; // i = 16
```

Concatenazione di stringhe

```
let myName = "pippo";  
console.log("ciao" + myName);  
> ciao pippo
```

Conversioni - funzioni utili

Description: The `parseInt()` function parses a string argument and returns an integer of the specified radix (the base in mathematical numeral systems).

Syntax: `parseInt(string, radix);`

Description: The `parseFloat()` function parses a string argument and returns a floating point number.

Syntax: `parseFloat(string);`

Conversioni

- Converta la stringa "42" in un numero (intero o float)

```
let num = Number("42");
```

- Converta il numero precedente in una stringa

```
num.toString();  
String(num)
```

- Regola il numero di cifre decimali

```
let pi = (3.141592).toFixed(2)  
> 3.14
```

Conversioni automatiche

```
var answer = "La risposta giusta e' ";
```

```
answer += 42
```

// 42 è convertito in stringa e concatenato

```
answer = "45" - 3; // 42 (conversione automatica)
```

```
answer = "45" + 3; // ????
```

Domanda

Ancora sulle variabili

```
const prefix = '06';
```

```
// read only
```

```
// case matters
```

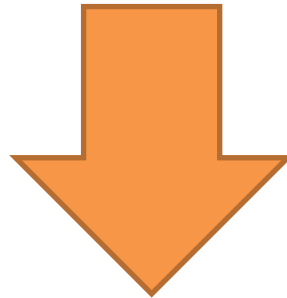
```
var programmazioneWeb;
```

```
var programmazioneweb;
```

Strict mode

`a = 5; // non uso let/var, dichiarazione implicita`

`NaN = true; // non produce errore ma non ha senso!`



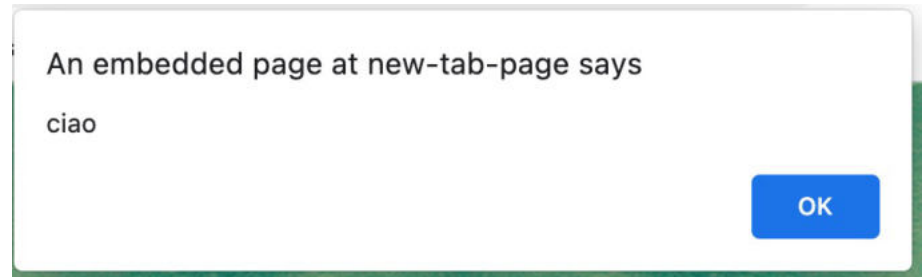
`"use strict"; // rende queste cose un errore`

- da mettere prima del resto del codice
- introdotto in ES5 per garantire retrocompatibilità
 - ad es. assegnamenti a proprietà non scrivibili o non esistenti lanciano un errore
- Usiamolo sempre per scrivere codice "moderno"

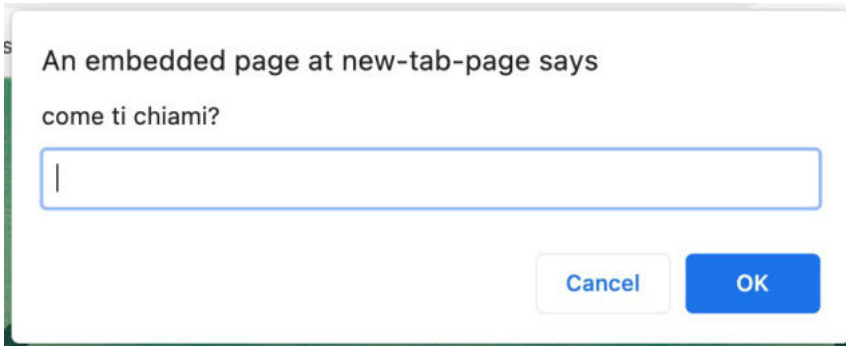
Approfondimento: https://www.w3schools.com/js/js_strict.asp

Interazione base

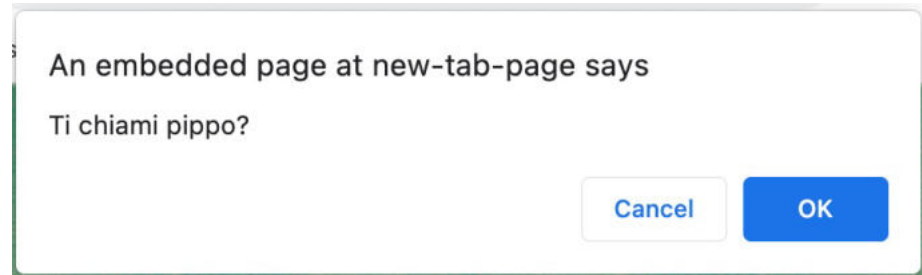
alert



prompt



confirm



- Funzioni utili all'interno del browser
- (vedremo più tardi che sono metodi dell'oggetto `window`)

Esercizio

Creare una calcolatrice per sommare due numeri

```
1  "use strict";  
2  
3  let n1, n2, somma;  
4  n1 = prompt("Inserisci il primo numero");  
5  n2 = prompt("Inserisci il secondo numero");  
6  
7  somma = Number(n1) + Number(n2);  
8  
9  alert("La somma fa " + String(somma));  
10
```

Esercizio 2

Quale di queste conversioni può essere *implicita*?

```
1  "use strict";
2
3  let n1, n2, somma;
4  n1 = prompt("Inserisci il primo numero");
5  n2 = prompt("Inserisci il secondo numero");
6
7  somma = Number(n1) + Number(n2);
8
9  alert("La somma fa " + String(somma));
10
```

Istruzioni di base

branch and loop

Operatori di comparazione

new!

<code>==</code> e <code>!=</code>	Uguale e non uguale
<code>></code> e <code>>=</code> e <code><</code> e <code><=</code>	Maggiore, maggiore o uguale, minore, minore o uguale
<code>===</code> e <code>!==</code>	Identico o non identico (stesso <i>dato</i> e <i>tipo</i>)

```
var a = "5"; // stringa contenente il numero 5
var b = 5; // numero 5
alert(a == b); // true
alert(a === b); // false
```

Operatori logici

&&	AND
	OR
!	NOT

> true && false

false

> (5 === 5) || false

true

Nullish coalescing operator (nuovo)

a ?? b

Se *a* è *null* o *undefined*, ritorna *b*.
Altrimenti torna *a*.

If, else

IF ternario

```
var status = (age >= 18) ? "adult" : "minor";
```

```
if (a == 5) {  
    alert("a e' 5");  
} else if (a > 5) {  
    alert("a e' maggiore di 5");  
} else {  
    alert("a e' minore di 5 (o non e' un numero!) ");  
}
```

Variabili truthy and falsy

Undefined, "", 0, false sono interpretati come "falsy"

```
var a = "";  
var b = "ciao"  
if (a) {alert("a e' falsy "); }  
if (b) {alert("b e' truthy"); }
```

Test

```
let a = 42, b;  
console.log(b ?? a == 42)  
  
console.log((b ?? a == 42) + 1)  
  
a ?? b ? a : b;
```

Switch

```
switch (expression) {  
    case label_1:  
        statements_1  
        [break;]  
    case label_2:  
        statements_2  
        [break;]  
    ...  
    default:  
        statements_def  
        [break;]  
}
```


Cicli for e while

```
/* ciclo for */  
for(let i = 0; i <= 10; i++) {  
    console.log(i);  
}
```

```
/* ciclo while */  
let i = -20;  
while(i > 0) {  
    console.log(i);  
    i++;  
}
```

Istruzioni di base

funzioni

Funzioni

Un modo pratico per **raggruppare** dei comandi e per **richiamare** più volte lo stesso codice

```
function calcolatrice() {  
    let n1, n2, somma;  
    n1 = prompt("Primo numero");  
    n2 = prompt("Secondo numero");  
    alert("La somma è " + (Number(n1) + Number(n2)));  
}
```

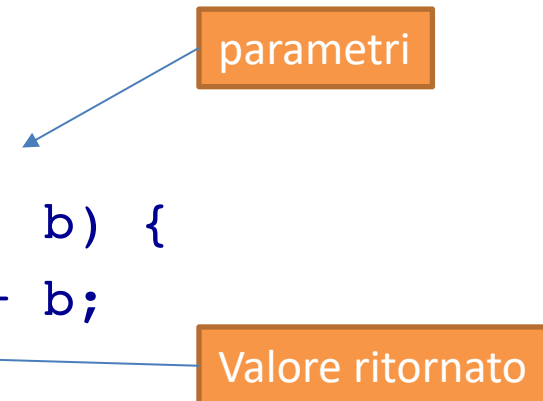
dichiarazione

```
calcolatrice();
```

invocazione

Funzioni con parametri e valori di ritorno

Le funzioni possono accettare dei parametri di ingresso e ritornare un valore in uscita



```
function somma(a, b) {
    let somma = a + b;
    return somma;
}
```


```
/* invochiamo una funzione e assegniamo il valore di
ritorno di una funzione aa una variabile */
let s = somma(3, 5); //s = 8
```

Lo scope: cos'è


- Lo scope è la **visibilità** di una variabile
 - La regione del nostro codice dove possiamo usare il nome della variabile/funzione
- Variabili locali definite dentro una funzione hanno lo scope relative al *blocco* della funzione stessa (**local scope**)
- Quando definiamo una variabile fuori da ogni funzione, è definita nel **global scope** e diventa visibile da ogni altro javascript che gira nella pagina
 - potenzialmente pericoloso! Interazioni non volute, spazio dei nomi ristretto!
 - namespace pollution (ovvero creazioni di variabili globali)

scope: esempi

```
// global scope
let a = 5;
function x() {
    return a + 1;
}
x(); // ritorna 6
a; // 5
```



```
// local scope
function x() {
    let a = 5;
    return a + 1;
}
x(); // ritorna 6
a; // undefined
```



Parametri

I parametri mancanti sono impostati a *undefined*

```
function somma(a, b) {  
    let somma = a + b;  
    return somma;  
}  
//b = undefined -> 3 + undefined = NaN  
let s = somma(3);
```

Possiamo impostare
valori di default

```
function somma(a, b = 1) {  
    let somma = a + b;  
    return somma;  
}  
let s = somma(3); //s = 4
```

Alternativa al valore di default
Usando operatori logici?

Debugging con Chrome

Impostiamo breakpoint

Controlliamo il flusso

The screenshot shows the Chrome DevTools interface with the following components:

- Elements Panel:** Shows the file structure with 'hello.html' and 'hello.js'.
- Source Panel:** Displays the code for 'hello.js'. A breakpoint is set at line 3, column 10, on the line `s = a + b;`. The code is:


```
function somma(a, b) {
  let s;
  s = a + b;
  return s;
}
somma(3, 2);
```
- Console Panel:** Shows the execution of the function `somma(3, 2);` with the result `5`.
- Watch Panel:** Shows 'No watch expressions'.
- Breakpoints Panel:** Shows a single breakpoint at 'hello.js:3' with the code `s = a + b;`.
- Scope Panel:** Shows the current scope with local variables:
 - `a: 3`
 - `b: 2`
 - `s: undefined`
 - `this: Window`
 - `Global: Window`
- Call Stack Panel:** Shows the call stack with two frames:
 - `somma` at `hello.js:3`
 - `(anonymous)` at `hello.js:7`

Valore delle variabili

Pila delle chiamate di funzioni

One function, one action

- Una funzione deve avere un nome descrittivo
 - Es. getName, runCalculator, fillResults, checkIsOnline
- La funzione deve fare esattamente una cosa
 - L'operazione descritta dal suo nome
 - Se fa più cose, probabilmente è utile suddividere il codice in due funzioni

Functional expression

```
let somma = function (a, b) {  
  let somma = a + b;  
  return somma;  
}  
let s = somma(3,2);
```

- Una **espressione funzionale** è creata quando l'esecuzione raggiunge quel punto
 - E' usabile da quel punto in poi
- Una **dichiarazione di funzione** può essere chiamata prima di quando sia definita

Passare funzioni ... a funzioni

```
function ask(question, yes, no) {
    if (confirm(question)) yes()
    else no();
}

ask(
    "Do you agree?",
    function() { alert("You agreed."); },
    function() { alert("You canceled the
execution."); }
);
```

Callback functions

Arrow functions

- Sistema più sintetico di specificare una funzione

```
let somma = (a, b) => a + b;
```

TEST

Riscrivere questa
funzione usando
arrow functions

```
function ask(question, yes, no) {  
  if (confirm(question)) yes()  
  else no();  
}  
  
ask(  
  "Do you agree?",  
  function() { alert("You agreed."); },  
  function() { alert("You canceled the  
execution."); }  
);
```

Cattiva programmazione

```
let data1;  
  
// Moltiplica data1 per 1.61803398875  
function c1() {  
    let x;  
    x = data1 * 1.61803398875;  
    return x;  
}  
  
data1 = 1000;  
let res = c1();
```

Evidenziare i problemi

Sistemiamo i problemi

```

1
2  /**
3   * Compute the golden rectangle dimensions.
4   * @param {number} shortSideLen the short side of the golden rectangle
5   * @returns {number} the long side of the golden rectangle
6   */
7  function getGoldenRectangle(shortSideLen) {
8      const phi = 1.61803398875; // the golden ratio
9      let longSideLen = shortSideLen * phi;
10     return longSideLen;
11 }
12
13 getGoldenRectangle(1000);|

```

JSDoc

Istruzioni di base

Oggetti

Oggetti

Un **oggetto** è una **lista di di coppie “proprietà” “valore”**, racchiuse in parentesi angolari { }

- Le proprietà possono essere solo stringhe o simboli *
- Un valore può essere un tipo primitivo, un altro oggetto o una funzione

```
var studente = {
  name: "Pierpaolo",
  age: 80,
  scores: [1,2,3],
  classes: {pw: 30, fi: 18}
};
```

da qui: JavaScript Object Notation (JSON)

stringa

intero

array

altro oggetto

Non esistono valori "privati"

* <https://javascript.info/symbol>

Oggetti

```
var studente = {}; // oggetto vuoto  
studente = new Object(); // stessa cosa
```

Creo oggetto

```
studente.voto = 30;
```

Aggiungo proprietà

```
console.log(studente.voto); // 30  
console.log(studente["voto"]); // stessa cosa
```

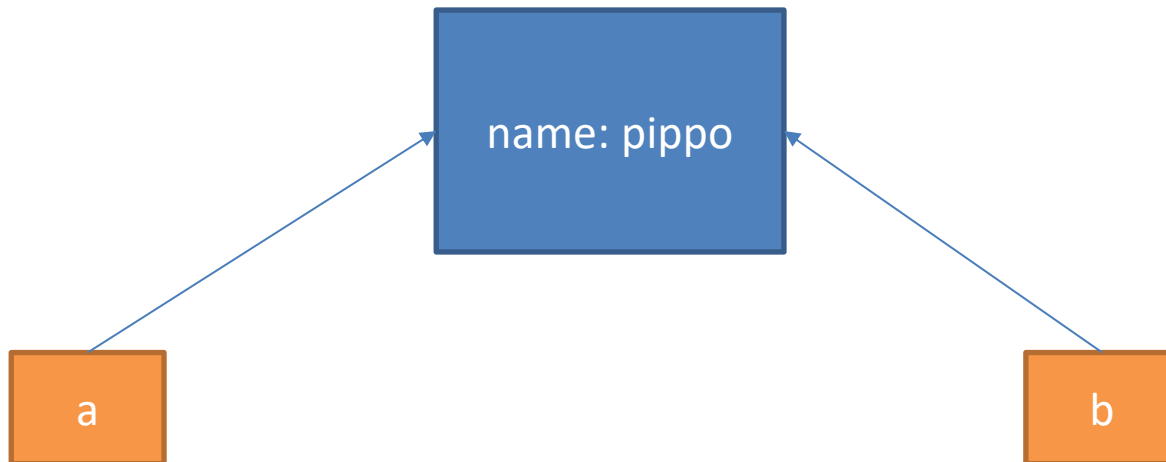
Accedo

```
delete studente.voto;  
console.log(studente.voto); // undefined
```

Rimuovo proprietà

Copiare un oggetto

```
let a { "nome": "pippo" };  
let b = a;
```



- A e B sono dei riferimenti allo stesso oggetto
 - Copiando a in b, copiamo solo il riferimento
 - Se facciamo `a.nome = "pluto"`, otterremo che anche `b.nome` è "pluto"

Test

```
let a = { "nome": "pippo" };  
let b = a;  
a == b
```

```
let a = {};  
let b = {};  
a == b
```

Garbage collection

- Tutte le variabili che creiamo occupano memoria che viene allocata dinamicamente
- Un algoritmo (*garbage collector*) capisce gli oggetti non più raggiungibili dallo script e rilascia la memoria

Vediamo un esempio:

```
let a = {"name": "pippo"}  
let b = a;
```

Codice

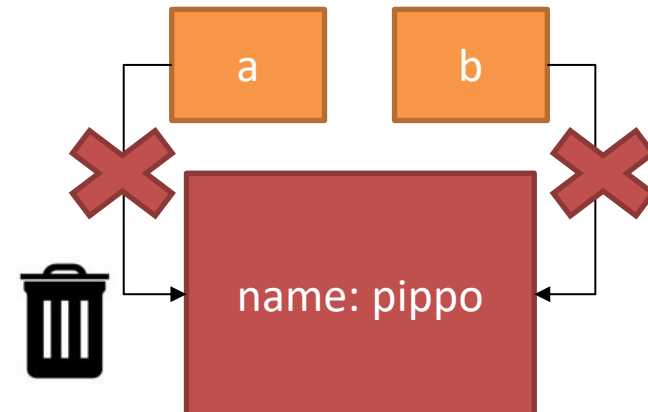
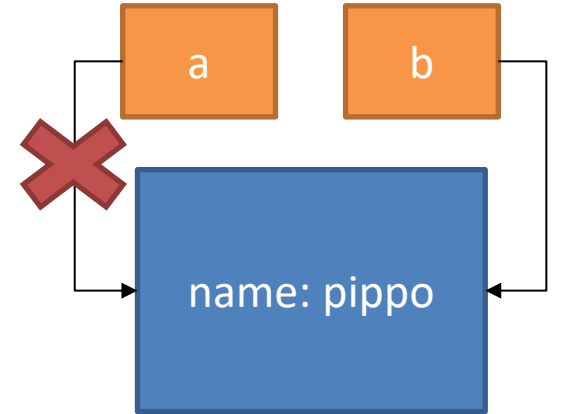
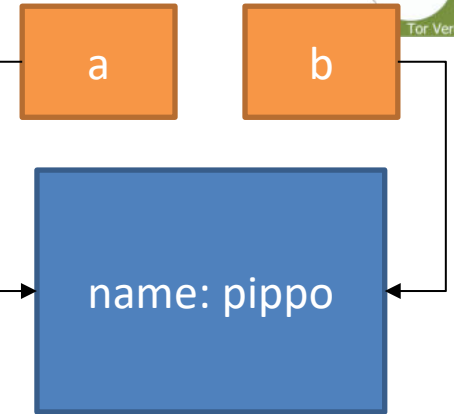
```
let a = {"name": "pippo"}  
let b = a;
```

```
a = null;
```

```
a = null;  
b = null;
```

Memoria

Variabili globali

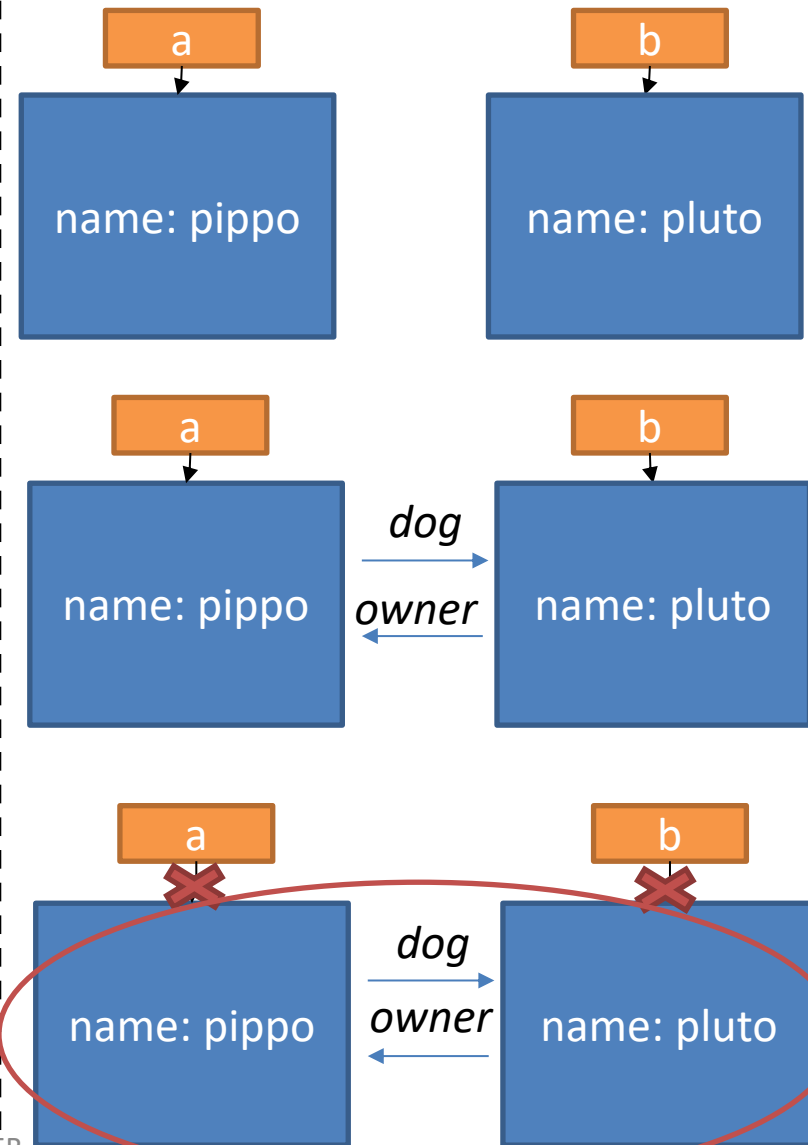


E' possibile avere dei memory leak

```
let a = {"name": "pippo"};  
let b = {"name": "pluto"}
```

```
a.dog = b;  
b.owner = a;
```

```
a = null;  
b = null;
```



Metodi

- Un oggetto puo' avere tra le sue proprietà anche delle funzioni che chiameremo *metodi*.

```
let a = {"name": "pippo"};  
a.saluta = function () {  
    alert("Ciao sono pippo");  
};
```

Dichiarazione

```
a.saluta();
```

Invocazione

This

- Può essere utile nei metodi riferirci ad altre proprietà dell'oggetto

```
let a = { "name": "pippo" };
a.saluta = function () {
    alert("Ciao sono " + this.name);
};
```

This è valutato a "call time"
(non quando è definita la funzione!)

This (binding)

```
let a = {"name": "pippo"};  
let b = {"name": "pluto"};  
  
function sayMyName () {  
    alert("Ciao sono " + this.name);  
};  
a.saluta = sayMyName;  
b.saluta = sayMyName;
```

Cosa fanno queste invocazioni?

```
a.saluta();  
b.saluta();  
sayMyName();
```

This e arrow function

- This nelle arrow functions si riferisce al outer scope (oggetto che le contiene)

```
let a = {"name": "pippo"};

a.saluta = function () {
    let x = () => alert("Ciao sono pippo");
    x(); // Ciao sono pippo
};
```

Esercizio

- Realizzare una calcolatrice con oggetti e metodi
 - Permettere l'inserimento di due numeri (funzione inserisci)
 - Implementare i metodi somma, sottrazione, moltiplicazione e di divisione

Costruttori

- Per creare oggetti uguali o simili possiamo usare delle funzioni

```
function User(name) {  
    this.name = name;  
    this.isAdmin = false;  
    // ritorna implicitamente this  
}
```

Costruttore

```
let user = new User("Pippo");  
alert(user.name); // Pippo  
alert(user.isAdmin); // false
```

Costruttori

Quando viene chiamato un costruttore con *new*:

- Viene creato un oggetto vuoto e assegnato a *this*
- Viene eseguita la funzione
- Viene ritornato *this*

Esperimento:
Che succede se non metto "new"?

Istruzioni di base

Array e stringhe

Tipi primitivi e metodi

- In JS esistono 7 tipi primitivi di dato
 - *string, number, bigint, boolean, symbol, null and undefined*
- *Questi tipi sono primitivi perché contengono un solo valore (ad es. "true")*
- *Hanno tuttavia dei metodi, come gli oggetti*
 - *Creati da un "object wrapper": un oggetto che viene creato e distrutto quando chiamiamo i metodi*

```
let v = "ciao";  
console.log(v.toUpperCase());
```

Altri metodi

- Tab on interactive console is your friend

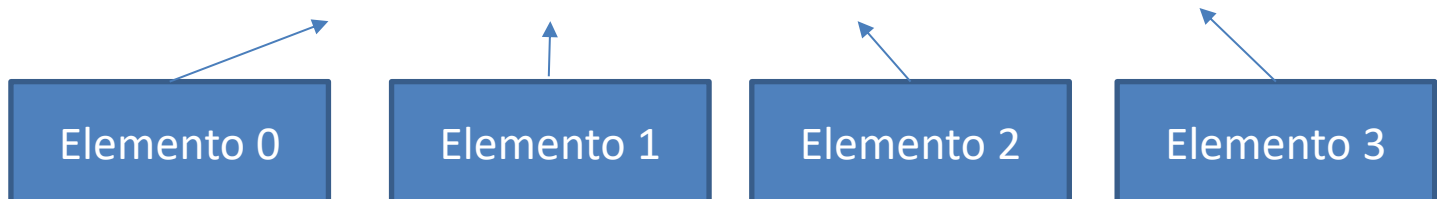
```
> s.
s.__defineGetter__
s.__defineSetter__
s.__lookupGetter__
s.__lookupSetter__
s.__proto__
s.hasOwnProperty
s.isPrototypeOf
s.propertyIsEnumerable
s.toLocaleString
s.anchor
s.big
s.blink
s.bold
s.charAt
s.charCodeAtAt
s.codePointAt
s.concat
s.constructor
s.endsWith
s.fixed
s.fontcolor
s.fontSize
s.includes
s.indexOf
s.italics
s.lastIndex0f
s.length
s.link
s.localeCompare
s.match
s.normalize
s.padEnd
s.link
s.padStart
s.repeat
s.replace
s.search
s.slice
s.small
s.split
s.startsWith
s.strike
s.sub
s.substr
s.substring
s.sup
s.toLocaleLowerCase
s.toLocaleUpperCase
s.toLowerCase
s.toString
s.trim
s.trimEnd
s.trimLeft
s.trimRight
s.valueOf
```


Array

- Contenitori di variabili, anche con tipi diversi
- Sono *oggetti* con proprietà numeriche e metodi/attributi per “maneggiarli” (more later)
- Ogni **elemento** dell’array ha un **indice** (che parte da 0)

Definizione

```
var myFirstArray = [5, "ciao", false, undefined];
```



Array

- Creare un array (metodi equivalenti)
 - `let arr = [element0, element1, ..., elementN];`
- Modificare un membro
 - `myFirstArray[0] = "nuovo valore";`
- Aggiungere un membro
 - `myFirstArray.push("ciao")` //aggiunge alla fine
 - `myFirstArray.unshift("ciao")` //aggiunge all'inizio
 - `myFirstArray[10] = "ciao";` // aggiunge al decimo posto (length sarà almeno 11)
- Rimuovere un membro
 - `myFirstArray.pop()` // rimuove ultimo elemento ritornandolo
 - `myFirstArray.shift()` // rimuove il primo elemento ritornandolo
 - `delete myFirstArray[10]` // rimuove l'elemento ma non sposta gli indici dell'array
- Lunghezza dell'array
 - `myFirstArray.length`
- Svuotare un array
 - `myFirstArray = []`
 - `myFirstArray.length = 0`

Array - esercizio

```
let giorno = [  
  "lunedì",  
  "martedì",  
  "mercoledì",  
  "giovedì",  
  "venerdì",  
  "sabato",  
  "domenica",  
];  
console.log("Il primo giorno della settimana è " + giorno[0]);  
> Il primo giorno della settimana è lunedì
```

In america la settimana parte da domenica. Trasformiamo l'array mettendo "domenica" all'inizio

Slicing

- `slice(start_index, end_index)` ritorna una porzione dell'array (non modifica l'array)

```
let myArray = ["a", "b", "c", "d", "e"];
myArray = myArray.slice(1, 4); // ritorna [ "b", "c", "d" ]
```

- `splice(index, n_elementi)` rimuove "n_elementi" partendo da "index", ritornandoli (modifica l'array).

```
let myArray = ["1", "2", "3", "4", "5"];
myArray.splice(3, 2); //ritorna ["4","5"]
// ora myArray e' ["1", "2", "3",]
```

Iteratori: For ... in, for ... of

- The **for...in** itera per le proprietà di un oggetto
- The **for...of** itera per gli elementi di un array/mappa/set (un "iterabile")

ECMAScript v6

```
let arr = [3, 5, 7];  
arr.foo = "hello";  
  
for (let i in arr) {  
    console.log(i); // logs "3", "5", "7", "foo"  
}  
  
for (let i of arr) {  
    console.log(i); // logs "3", "5", "7"  
}
```

Array: loop, join e search

```
let colors = ['red', 'green', 'blue'];
```

```
// loop
```

```
colors.forEach(function(color) {  
    console.log(color);  
});
```

```
// join
```

```
let list = myArray.join(" - "); // "red - green - blue"
```

```
// search
```

```
let a = ['a', 'b', 'a', 'b', 'a'];  
console.log(a.indexOf('b')); // 1
```

Attenzione agli indici!

```
let a = ["a", "b", "c"];
```

```
a.length;
```

ritorna 3

```
delete a[0];
```

[undefined, 'b', 'c']

```
a.length;
```

ritorna sempre 3

```
a[10] = "d";
```

```
a.length;
```

ritorna 11

Map (array method)

- Metodo che serve a "convertire" (mappare) un array in un altro
 - Item: i-simo oggetto dell'array
 - Index: indice dell'oggetto (partendo da 0)
 - Array: l'array
- Esempio: dato un array di stringhe, generiamo un array contenente quanto sono lunghe

```
let a = ["pippo", "pluto", "paperino"];
```

```
a.map( (item, index, array) => item.length);
```

```
// ritorna [5, 5, 8]
```


Reduce

- Metodo che serve a calcolare un singolo valore dall'array

```
arr.reduce(  
    function(accumulator, item, index, array){  
        // ...  
    },  
    [initial]  
);
```

- Accumulator è il risultato della chiamata a funzione precedente, o initial se è la prima volta.

Reduce (esempio)

- Sommiamo tutti gli elementi di un array

```
[1, 2, 3].reduce((ac, item) => ac + item);
```

```
// torna 6
```

Stringhe

```
let s = "Ciao a tutti"; // creiamo una stringa
```

```
s = "ciao a \n  
tutti"; // se è lunga possiamo andare a capo
```

```
s.indexOf(" a "); // ritorna 4 (prima occorrenza)  
s.slice(1); // ritorna "iao a tutti" (non modifica la stringa)  
s.trim(); // leva whitespaces ad inizio e fine stringa  
s.charAt(1); // ritorna "i" (carattere a indice 1)  
s.toUpperCase(); // ritorna la stringa in maiuscolo  
s.toLowerCase(); // ritorna la stringa in minuscolo
```

Stringhe - sostituzione

```
//Cambia la prima occorrenza di "ciao" in  
"bye"
```

```
"ciao ciao".replace("ciao", "bye");
```

```
// se voglio cambiare tutte le occorrenze devo  
usare un'espressione regolare e applicarla  
globalmente (g)
```

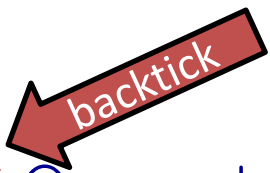
```
"ciao ciao".replace(/ciao/g, "bye")
```

Stringhe - formattazione

```
// template string
```

```
let lessonNumber = 3;
```

```
console.log(`Questa è la lezione  
numero ${lessonNumber} di  
javascript`);
```



Introdotta con ECMA6
Prima si usava sprintf

Esercizio

```
let names = ['mario', 'giovanna', 'pippo'];
```

Modificare l'array per:

1. Convertire i nomi in maiuscolo
2. Aggiungere "Dr. " prima del nome
3. Calcolare chi ha il nome più lungo

Array e Stringhe come oggetti

Array e strings sono oggetti

Questi metodi sono equivalenti:

```
let a = []
```



```
let a = new Array()
```

```
let a = Array()
```

Stesso vale per le stringhe:

```
let s = new String("Ciao a tutti");
```

```
let s = "ciao a tutti"
```

* Oltre a essere più corto, crea meno problemi – approfondimento:

<http://bonsaiden.github.io/JavaScript-Garden/#array.constructor>

Esercizio

- Realizzare un modello di dati per un sito di e-commerce con:
 - Un array contenente i prodotti (id, descrizione, costo, disponibilità)
 - Un oggetto "carrello" con metodi aggiungi, rimuovi, e guarda



Istruzioni di base

Built-in objects

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Date

- L'oggetto built-in "Date" ha metodi e costanti per le date (js non ha il tipo primitivo "data")

```
today = new Date() // data di oggi
var Xmas95 = new Date("December 25, 1995 13:30:00")
Xmas95 = new Date(1995, 11, 25)
var Xmas95 = new Date(1995, 11, 25, 9, 30, 0)
```

```
Xmas95.getMonth() // ritorna 11
Xmas95.getFullYear() //ritorna 1995.
getTime() // ritorna i millisecondi dal 1-1-1970
```

Typeof e instanceof

- `typeof true;` `// returns "boolean"`
- `typeof 62;` `// returns "number"`

- `var theDay = new Date(1995, 12, 17);`
- `theDay instanceof Date;` `// true`

Math

- L'oggetto built-in "Math" ha metodi e costanti per operazioni matematiche

Funzioni/costanti	Descrizione
sin(), cos(), tan()	Funzioni trigonometriche
pow(), exp(), expm1(), log10(), log1p(), log2()	Funzioni esponenziali
min(), max()	Ritornano min o max di una lista
random()	Ritorna un numero casuale tra 0 e 1
PI	costante per pi greco
round(), fround(), trunc()	Arrotondamenti e troncamenti

Esempio:

```
Math.random() // 0.932132321
```

JSON

- E' utile convertire oggetti in stringhe (e viceversa), ad esempio per importare/passare dati a un server

```
JSON.stringify(object)
```

```
JSON.parse(objectString) ;
```

Window

- L'oggetto window rappresenta la finestra del browser

Metodi piu' usati	Descrizione
alert(), confirm(), prompt()	Visualizza un messaggio, chiede conferma con finestra di dialogo, chiede un testo
open(), close()	Apri e chiude una finestra
print()	Stampa
ScrollTo()	Esegue lo scrolling fino a delle coordinate
setInterval(), clearInterval()	Richiama (o annulla) una funzione ogni TOT tempo
setTimeout(), clearTimeout()	Imposta (o annulla) un timer

Funzioni dell'oggetto window

Oltre a *alert*, *confirm*, *prompt*, l'oggetto window ha altre funzioni ("metodi") utili:

- **setTimeout**(*funzione_da_chiamare*, *time*):
richiama la funzione scelta dopo *time* millisecondi
- **setInterval**(*funzione_da_chiamare*, *time*): richiama la funzione **ciclicamente** dopo *time* ms
- **clearTimeout** e **clearInterval** interrompono le funzioni precedenti
(vedere guida: http://www.w3schools.com/jsref/obj_window.asp)

Eccezioni: a che servono?

- Indicano che qualcosa è andato storto...
 - es. `jnkdfsnjkfd (); // ReferenceError: jnkdfsnjkfd is not defined`
- Un eccezione può essere qualunque tipo di dato (oggetto, stringa, numero...)
- Il frammento di codice “lancia” un’eccezione (**throw**), che può essere gestita (**catch**)
 1. Interrompe la normale esecuzione
 2. cerca una routine in grado di risolvere il problema (**catch**)
 3. se “gestita”, il flusso continua da dopo il blocco “catch”

Eccezioni: Esempio

```
function getMonthName(monthId) {
    if (monthId == 1) { return "Gennaio";}
    else if (monthId == 2) { return "Febbraio";}
    /* ... */
    else if (monthId == 12) { return "Dicembre";}
    else {
        throw "Il mese non e' valido";
    }
}
```

C'e' un problema, lancio l'eccezione

```
function f(myMonth) {
    try {
        monthName = getMonthName(myMonth);
    }
    catch (e) {
        monthName = "unknown";
    }
    finally {
        // eseguita in ogni caso (ad es. chiudi un file)
    }
}
```

Gestisco l'eccezione

Error object

Struttura dati “errore generico” per l’eccezione.

Due proprietà:

1. name: errore sintetico (“DOMException”)
2. message: descrizione verbosa dell’errore

```
throw (new Error( 'The message' ) );
```

Esistono Errori più specifici (*ReferenceError*, *URIError* ...)

lista completa su:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error#Error_types

Approfondiamo le funzioni

Closure

Var e Let

- lo *scope* di *var* è il *functional block* più vicino
- lo *scope* di *let* è l'*enclosing block* più vicino

```
for(let i=0; i<2; i++) {  
    console.log(i);  
}  
console.log(i); // i is not defined
```

ES6

```
for(var i=0; i<2; i++) {  
    console.log(i);  
}  
console.log(i); // i = 2
```

Se ci dimentichiamo di dichiarare una variable, diventerà una proprietà dell'oggetto window

Funzioni e oggetti

- Possiamo definire funzioni dentro altre funzioni
- La funzione “nested” può accedere allo scope della funzione che la include (oltre che allo scope globale)

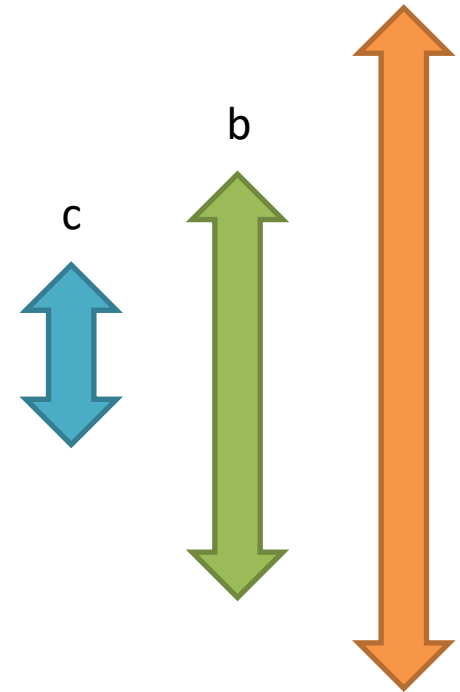
```
function x(p1) {
  let a = p1; //outer scope di y
  function y() {
    return a*2;
  }
  return y();
}
```

funzione “nested”
(inner function)

Scope e funzioni nested

```
function molto_fuori() {
  let a = 5;
  function fuori() {
    let b = 6;
    function dentro() {
      let c = 7;
      console.log(a,b,c);
    }
    return dentro();
  }
  return fuori();
}
molto_fuori();
```

Scope



L'inner function può accedere allo scope delle outer functions
l'outer function non può accedere allo scope delle inner functions

Funzioni che ritornano funzioni

```
function multisum(p1, a, b) {
    let x = p1;
    function sum(a, b) {
        return x * (a + b);
    }
    return sum(a,b);
}
```

`multisum(10, 1,2)` ← torna 30

La funzione "multisum" ritorna l'output di "sum", ovvero ritorna un numero

```
function multisum(p1) {
    let x = p1;
    return function sum(a, b) {
        return x * (a + b);
    }
}
```

`multisum(10);` ← torna una funzione

`multisum(10)(1,2)` ← torna 30

Closure

```
function multisum(p1) {  
  let x = p1;  
  return function sum(a, b) {  
    return x *(a + b);  
  }  
}
```

Ambiente (scope
outer function)

Inner function (lo
scope “si chiude” su
quello del padre)

Esempio di closure

```
function salutatore(name) {  
    let text = 'Ciao' + name; // Local variable  
    let diCiao = function() { alert(text); }  
    return diCiao;  
}  
  
let s = salutatore('Lorenzo');  
s(); // alerts "Ciao Lorenzo"
```

"s" non memorizza solo il return della funzione "salutatore" (che è una funzione), ma anche il suo scope esterno (ad es la variabile "text")

Perchè le closure sono utili?

```
function counter() {
  let a = 0;
  return {
    inc: function() { ++a; },
    dec: function() { --a; },
    get: function() { return a; },
    reset: function() { a = 0; }
  }
}
```

Definizione



```
let c = counter();
c.inc();
```

Utilizzo

- Ho quindi metodi o attributi privati
 - Se faccio “c.a” ottengo un errore
- Simulo object oriented programming
 - "object data privacy": separiamo interfaccia da implementazione

Impostiamo un programma javascript

mioprogramma.js

```
let a = 0;  
let b = 0;  
function pippo(x,y) {  
    // qui mettiamo del codice  
    return x*y;  
}  
// ... altro
```

Problema

Ho scritto a,b e pippo nel global scope!
sono l'unico a usare questi nomi?

Soluzione

Mettiamo tutto il codice in una funzione!
Invochiamo questa funzione all'avvio

Independently Invoked Functional Expression (IIFE)

mioprogramma.js

```
(function() {  
    let a = 0;  
    let b = 0;  
    function pippo(x,y) {  
        // codice di esempio  
        return x*y;  
    }  
})();
```

Funzione ANONIMA invocata immediatamente

Tecnica molto usata

```

/*!
 * jQuery JavaScript Library v2.1.4
 * http://jquery.com/
 *
 * Includes Sizzle.js
 * http://sizzlejs.com/
 *
 * Copyright 2005, 2014 jQuery Foundation, Inc. and other contributors
 * Released under the MIT license
 * http://jquery.org/license
 *
 * Date: 2015-04-28T16:01Z
 */

(function( global, factory ) {

```



Quiz

Da
intervista
vera!

```
(function() {  
    let a = b = 5;  
})();
```

```
console.log(b);  
//quanto stampa?
```

Approfondiamo gli oggetti

Costruttori, prototipi ed ereditarietà

...con i 3 porcellini



This

- Un oggetto può avere come proprietà una funzione
- La parola chiave **this** usata dentro la funzione indica l'oggetto che la contiene
 - Dipende dal contesto, la stessa funzione può indicare come "this" oggetti diversi

Esempio:

```
var studente = {  
  name: "pippo",  
  getName : function() {  
    return this.name;  
  }  
}
```

Attenzione al contesto!

E' valutato a call-time

```
let x = 9;  
let module = {  
  x: 81,  
  getX: function() { return this.x; }  
};
```

```
module.getX(); // 81
```

```
let getX = module.getX;  
getX(); // quanto ritorna?
```

fuori da un oggetto (scope
globale)

this === window

Ripasso: costruttore

- Posso creare un oggetto normalmente ...

```
let jimmy = {name: "Jimmy", color: "pink", age: 0};  
let timmy = {name: "Timmy", color: "pink", age: 0};
```

- Oppure posso usare una funzione che imposta le proprietà dell'oggetto

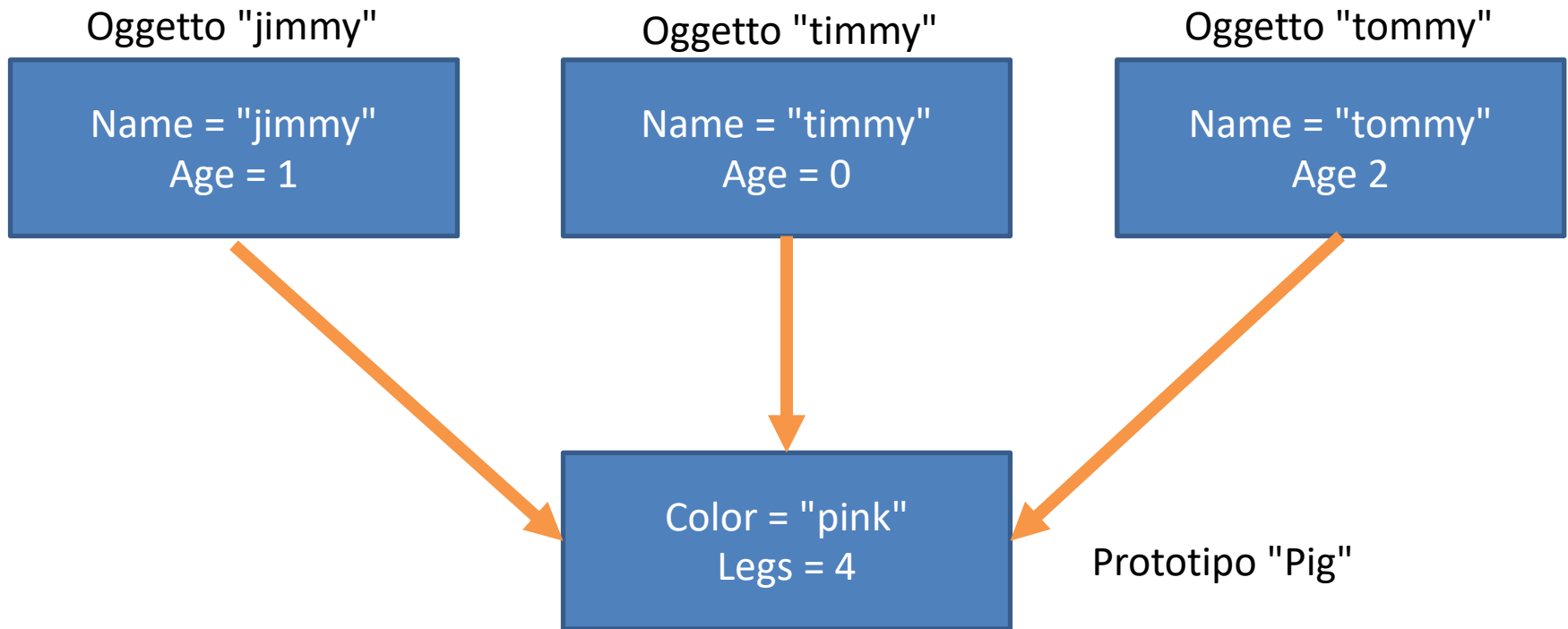
```
function Pig(name, color) {  
  this.name = name;  
  this.color = color;  
  this.age = 0;  
}
```

```
var tommy = new Pig("Tommy", "pink");
```

- Questa funzione si chiama "costruttore"
 - E' una funzione normalissima, ma la usiamo per "costruire" un oggetto

Prototipi di oggetto

- In JS gli oggetti hanno un prototipo, che è un altro oggetto da cui eredita tutte le proprietà

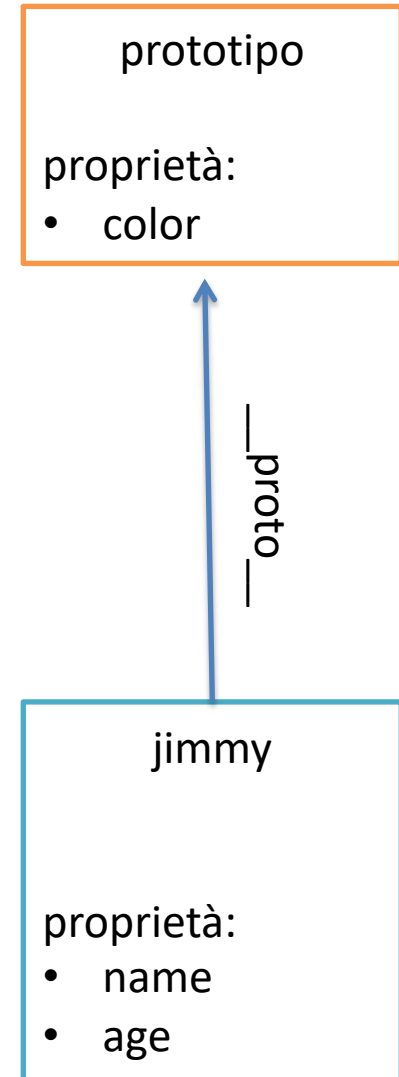


Possiamo vedere il prototipo di un oggetto scrivendo:
`myObject.__proto__`

Oggetti: Proprietà di base e ereditate

- Quando chiamiamo una proprietà di un oggetto:
 - prima si cerca tra le proprietà dell'oggetto
 - poi tra le proprietà del prototipo
 - poi tra le proprietà del prototipo del prototipo (etc) [**Prototype chain**]
- Esempio
 - `jimmy.name` → trova la proprietà nell'oggetto
 - `jimmy.color` → trova la proprietà nel prototipo
- Possiamo vedere se la proprietà è dell'oggetto o del prototipo con:

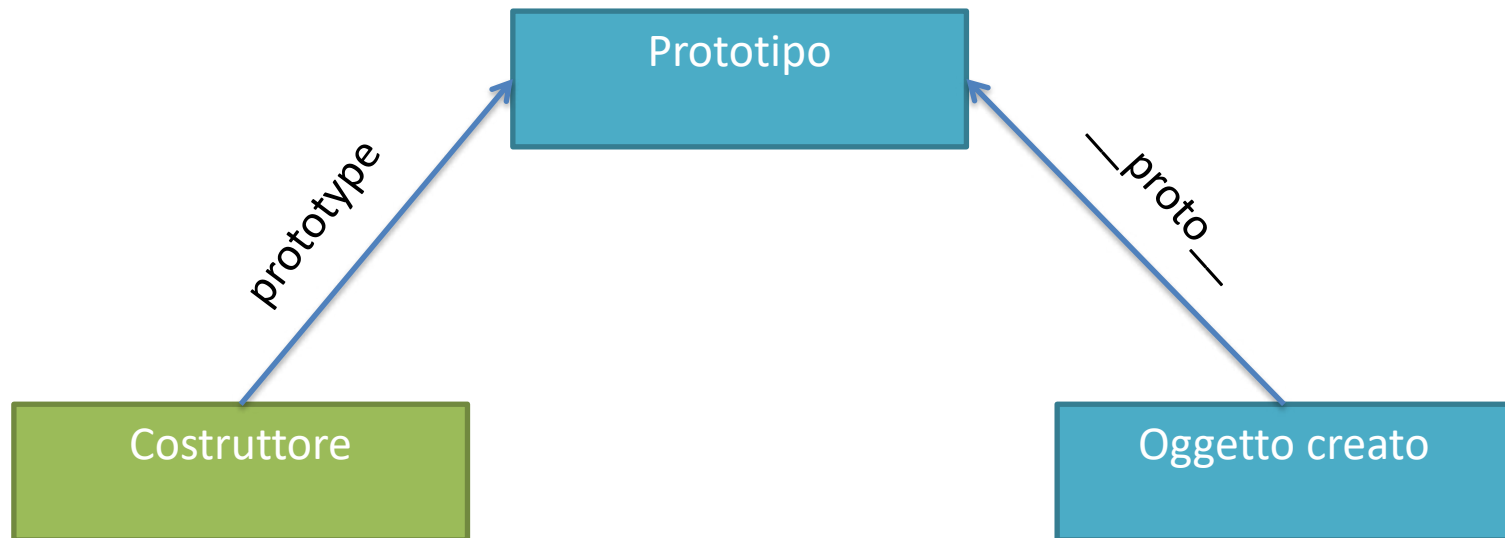

```
jimmy.hasOwnProperty("color"); // false
jimmy.__proto__.hasOwnProperty('color'); // true
```
- Quindi possiamo ridefinire (override) alcune proprietà del prototipo



Prototipi di funzione

- Ogni funzione ha la proprietà “**prototype**” il cui valore è un oggetto
- Scrivendo `Pig.prototype.color = "pink"` assegniamo una proprietà a quell'oggetto
- Tutti gli oggetti creati con questo costruttore, avranno come prototipo il prototipo della funzione

Costruttori, prototipi e proprietà



Funzione



Istanza oggetto

```
function Pig(name, age) {  
    this.name = name;  
    this.age = age;  
}  
Pig.prototype.color = "pink";  
  
let tommy = new Pig("Tommy", 2);
```

Cosa succede?

1. Viene creato un nuovo oggetto vuoto
2. Viene passato al costruttore (`function Pig`), in modo che ci possa riferire con *"this"*
3. Il costruttore setta le proprietà dell'oggetto
4. Il costruttore imposta:
prototipo dell'oggetto creato = prototipo della funzione
`Pig.prototype` → `tommy.__proto__`

Prototipi

prototipi di funzione

E' l'istanza di un oggetto che diventerà il prototipo per tutti gli oggetti creati usando la funzione come costruttore

`NomeFunzione.prototype`

prototipi di oggetto

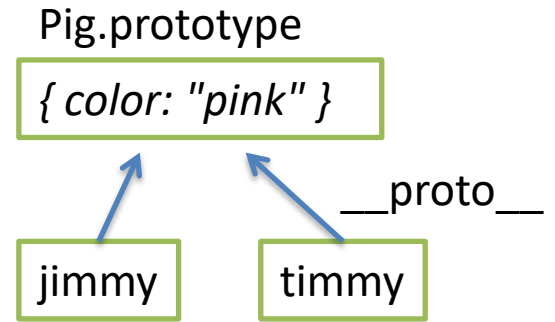
E' l'istanza dell'oggetto dal quale l'oggetto è ereditato

`NomeOggetto.__proto__`

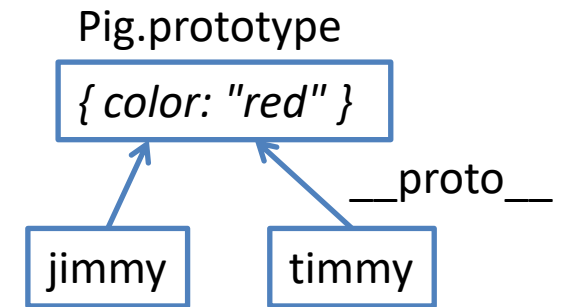
```
function Pig(name, age) {
  this.name = name;
  this.age = age;
}
Pig.prototype.color = "pink";

let jimmy = new Pig("Jimmy", 1);
let timmy = new Pig("Timmy", 2);

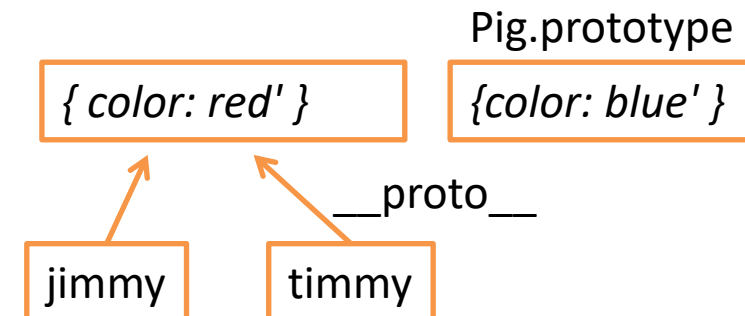
jimmy.color;
timmy.color;
```



```
Pig.prototype.color = "red";
jimmy.color;
timmy.color;
```



```
Pig.prototype = {color: "blue"};
jimmy.color;
timmy.color;
```



Esercizio

- Far direi il proprio nome ai 3 porcellini
 - implementare `pigX.sayName()`



Cosa scegliereste?

```
function Person(name) {
    this.name = name
    this.sayHi = function() {
        return 'Hi, I am ' + this.name
    }
}
```

Se creiamo molte persone, usando l'eredità instanzio solo una volta la funzione "sayHi"

```
function Person(name) {
    this.name = name
}
Person.prototype.sayHi = function() {
    return 'Hi, I am ' + this.name
}
```

Bind

- Il metodo bind ci permette di definire chi è il “this” per una funzione
 - metodo di una funzione → la funzione è un oggetto! (Function object)

```
let a = {id: 10};
```

```
let x = function() {return this.id}
```

```
let w = x.bind(a);
```

```
x(); // ritorna undefined
```

```
w(); //ritorna 10
```

Apply e Call: impostare il “this”

- **apply** per chiamare una funzione impostando un certo this e passando gli argomenti come array
- **call** come apply, ma gli argomenti sono passati esplicitamente

```
myFunction.apply(myObject, ["Susan", "teacher"]);
myFunction.call(undefined, "Claude", "mathematician");
```

valore di “this”

New e apply

```
function Car(maker, model, year) {  
    this.maker = maker;  
    this.model = model;  
    this.year = year;  
}  
let mycar = new Car("FIAT", "500", 1936);  
  
// oppure ...  
let new_car = new Object()  
Car.apply(new_car, ["FIAT", "500", 1936]);
```

Copiare un oggetto

```
let a = {id: 10};
```

```
let b = a;
```

```
b.id = 11; // anche a.id e' 11
```

PERICOLOSO!

- Vogliamo duplicare un oggetto? Dobbiamo farlo a mano!
- E il prototipo?

Global objects

A global object is an object that always exists in the global scope

In a web browser, when scripts create global variables defined with the `var` keyword, they're created as members of the global object (not in NodeJS)

Global objects:

- Browser -> `window`
- Nodejs → `global`
- Webworkers -> `WorkerGlobalScope`

```
var foo = "foobar";  
foo === window.foo; // Returns: true
```

https://developer.mozilla.org/en-US/docs/Glossary/Global_object