

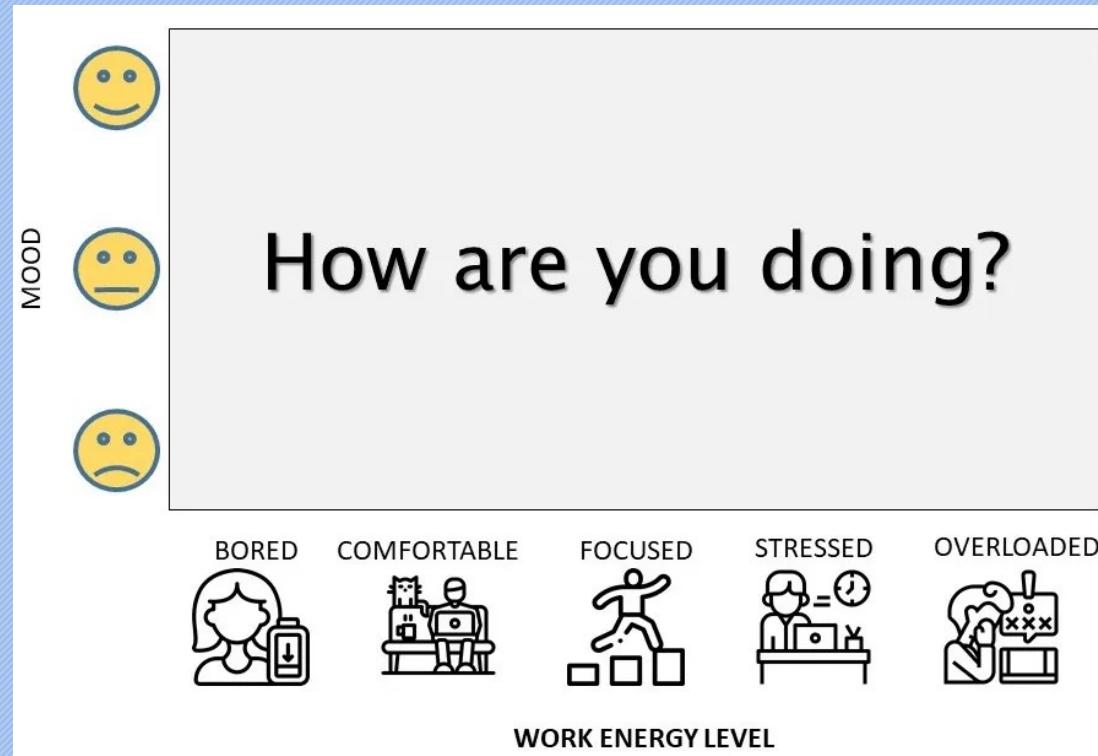
INTRODUZIONE A JAVASCRIPT

MININNI DAVIDE
SOFTWARE ENGINEER - SENIOR ANALYST
FINCONS S.P.A.

05/2024



CHECK-IN



PROGRAMMA DEL CORSO - 1



INTRODUZIONE

- CARATTERISTICHE E STORIA DI JAVASCRIPT

JAVASCRIPT BASICS - 1

- JAVASCRIPT NEL BROWSER
- STATEMENTS E VARIABILI
- DATA TYPES
- OPERATORI
- CONTROL FLOW

PROGRAMMA DEL CORSO - 2



JAVASCRIPT BASICS - 2

- FUNZIONI
- PROTOTIPO
- OGGETTI
- ARRAY
- DATE
- MAP / SET
- BROWSER
 - DOCUMENT
 - EVENTS
 - FORMS

PROGRAMMA DEL CORSO - 3



JAVASCRIPT ADVANCED

- ASYNC / AWAIT
- PROMISES
- JSON
- NETWORK REQUESTS
- TYPESCRIPT
- FRAMEWORKS
- WEB-COMPONENTS
- ...

STRUMENTI - 1



- STANDARD ECMASCIPT: <https://ecma-international.org/publications-and-standards/standards/ecma-262/>
- JAVASCRIPT SU MDN: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- TUTORIAL CON ESEMPI: <https://www.w3schools.com/js/>
- CORSO COMPLETO JS: <https://javascript.info/>
- CORSI BASE/AVANZATI (FREEMIUM):
<https://www.codecademy.com/catalog/language/javascript>

STRUMENTI - 2



- MOZILLA DEVELOPER NETWORK:
<https://developer.mozilla.org/en-US/>
- IDE (INTEGRATED DEVELOPMENT ENVIRONMENT):
 - VISUAL STUDIO CODE: <https://code.visualstudio.com/>
 - JETBRAINS WEBSTORM: <https://www.jetbrains.com/webstorm/>
- DEVELOPER'S CONSOLE (F12 IN BROWSER)
- JAVASCRIPT PLAYGROUND: <https://playcode.io/javascript>

INTRODUZIONE - 1



JAVASCRIPT, SPESSO ABBREVIATO IN JS, È UN LINGUAGGIO DI PROGRAMMAZIONE, NONCHÈ UNA TECNOLOGIA FONDAMENTALE DEL WEB, INSIEME A HTML E CSS.

IL 98,9% DEI SITI WEB UTILIZZA JAVASCRIPT SUL LATO CLIENT PER LA GESTIONE DEL COMPORTAMENTO DELLA PAGINA WEB.

I PROGRAMMI IN JS SONO CHIAMATI «*SCRIPT*»: POSSONO ESSERE SCRITTI ALL'INTERNO DI UNA PAGINA HTML E SONO ESEGUITI AUTOMATICAMENTE AL CARICAMENTO DELLA PAGINA.

INTRODUZIONE - 2

PERCHÉ JAVASCRIPT?

- PERFETTAMENTE INTEGRATO CON HTML/CSS
- «*DOING SIMPLE THINGS SIMPLY*»
- SUPPORTATO DA TUTTI I BROWSER E ABILITATO PER DEFAULT
- PUÒ ESSERE ESEGUITO SU ANCHE SU SERVER (NODE.JS)



COSA SI PUÒ FARE IN JAVASCRIPT NEL BROWSER:

- CAMBIARE IL CONTENUTO DI UNA PAGINA WEB;
- MODIFICARE LO STILE DI UNA PAGINA WEB;
- PERMETTERE L'INTERAZIONE UTENTE (CLICK, KEYPRESS..);
- INVIARE RICHIESTE A SERVER REMOTO;
- ...

«JAVASCRIPT WAS CREATED TO MAKE WEB PAGES ALIVE»

INTRODUZIONE - 4



COSA NON SI PUÒ FARE IN JAVASCRIPT NEL BROWSER:

- ACCEDERE ARBITRARIAMENTE A FILE SULL'HARD DISK;
- ACCEDERE ARBITRARIAMENTE A POSIZIONE, FOTOCAMERA, MICROFONO;
- TRASMETTERE ARBITRARIAMENTE INFORMAZIONI TRA PAGINE;
- RICEVERE DATI ARBITRARIAMENTE DA ALTRI DOMINI
- ...

«USER'S SAFETY COMES FIRST»

INTRODUZIONE - 5



JAVASCRIPT È UN LINGUAGGIO

- DI ALTO LIVELLO
- INTERPRETATO / COMPILATO JIT
- STANDARDIZZATO (ECMASCRIPT 1997-2023)
- IMPERATIVO E STRUTTURATO
- DEBOLMENTE TIPIZZATO E DINAMICO
- ORIENTATO AGLI OGGETTI TRAMITE PROTOTIPO
- FUNZIONALE

CARATTERISTICHE DI JAVASCRIPT - 1



JAVASCRIPT È UN LINGUAGGIO DI ALTO LIVELLO, OVVERO È PROGETTATO PER PERMETTERE ALL'UTENTE DI SVILUPPARE CODICE SENZA DOVER CONOSCERE I DETTAGLI DELL'HARDWARE SU CUI VERRÀ ESEGUITO IL PROGRAMMA.

IL SOFTWARE CHE ESEGUE IL CODICE JS È DETTO «JAVASCRIPT ENGINE». OGNI BROWSER HA IL SUO:

- CHROME: V8
- MOZILLA: SPIDERMONKEY
- SAFARI: WEBKIT (JAVASCRIPTCORE)

CARATTERISTICHE DI JAVASCRIPT - 2

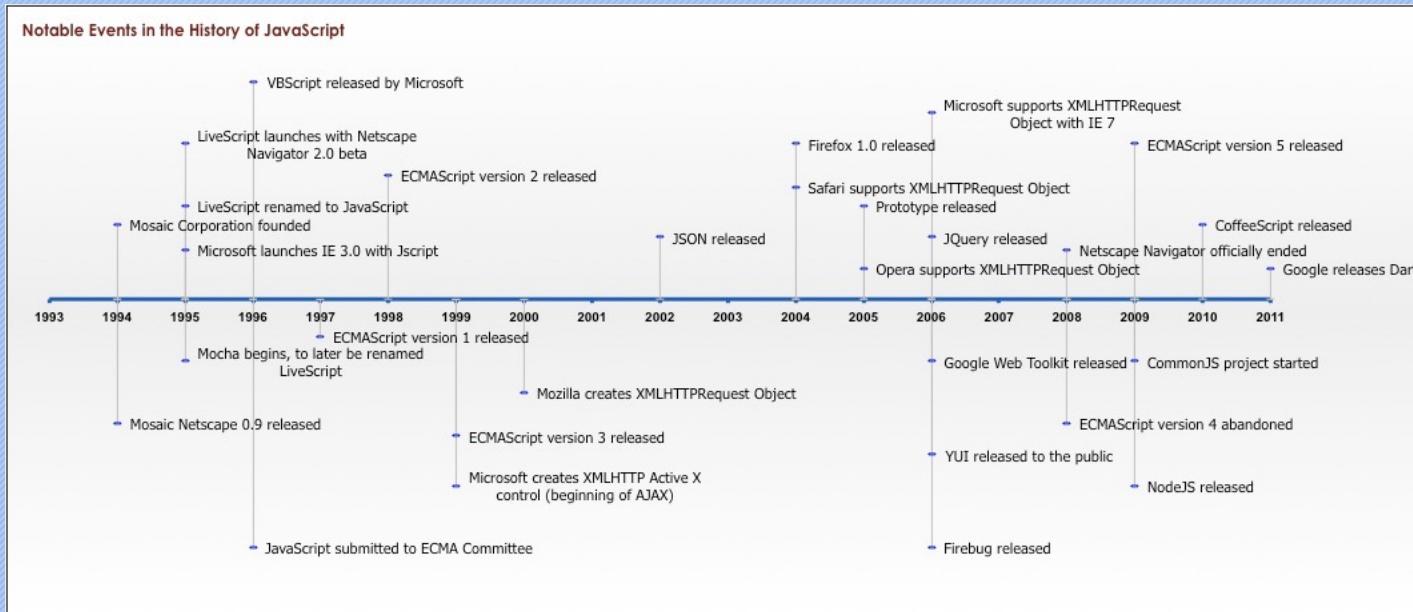


PER ESSERE ESEGUITO, IL CODICE DEVE ESSERE «CONVERTITO» IN LINGUAGGIO MACCHINA TRAMITE COMPILER/INTERPRETER.

JAVASCRIPT È ATTUALMENTE UN LINGUAGGIO COMPILOTATO «JUST-IN-TIME»; IN ORIGINE ERA INTERPRETATO.

QUAL È LA DIFFERENZA?

CARATTERISTICHE DI JAVASCRIPT - 2.1

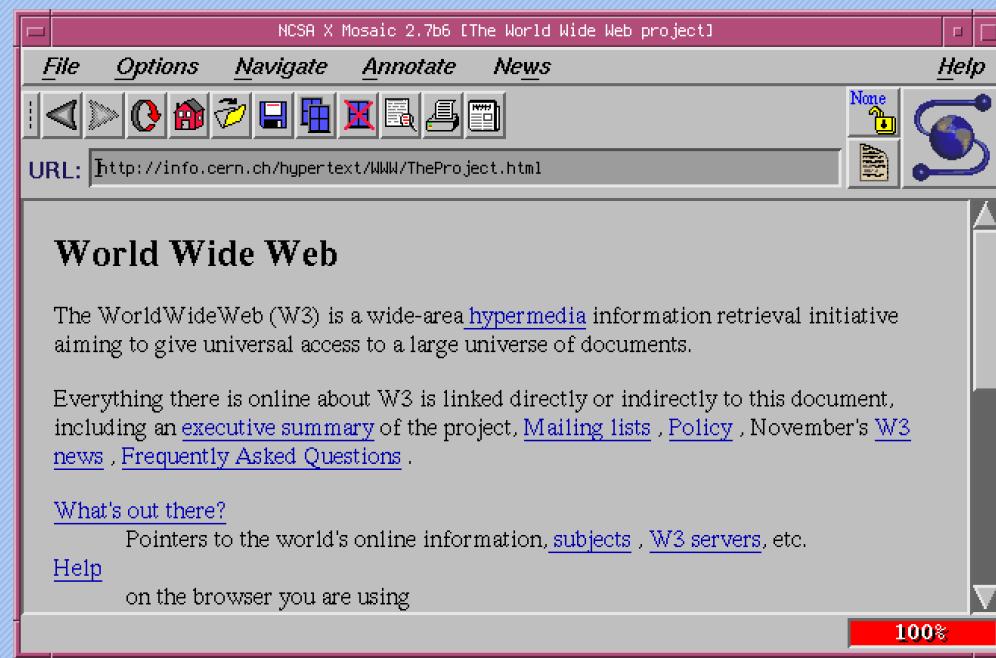


CARATTERISTICHE DI JAVASCRIPT - 2.2



- 1993: VIENE RILASCIATO «MOSAIC», IL PRIMO BROWSER CON INTERFACCIA GRAFICA.
- 1994: GLI SVILUPPATORI DI «MOSAIC» CREANO L'AZIENDA NETSCAPE E RILASCIANO «NETSCAPE NAVIGATOR».
LE PAGINE WEB SONO STATICHE, SENZA POSSIBILITÀ DI AGGIUNGERE COMPORTAMENTI DINAMICI UNA VOLTA CARICATE DAL BROWSER.

CARATTERISTICHE DI JAVASCRIPT - 2.3



CARATTERISTICHE DI JAVASCRIPT - 2.4



- 1995: NETSCAPE ASSUME BRENDAN EICH PER CREARE UN NUOVO LINGUAGGIO CON L'OBBIETTIVO DI RENDERE 'VIVE' LE PAGINE WEB.

IL NUOVO LINGUAGGIO E IL SUO INTERPRETER PRENDE IL NOME DI «MOCHA», POI «LIVESCRIPT», INFINE «JAVASCRIPT».

 Company Press Relations

NETSCAPE AND SUN ANNOUNCE JAVASCRIPT, THE OPEN, CROSS-PLATFORM OBJECT SCRIPTING LANGUAGE FOR ENTERPRISE NETWORKS AND THE INTERNET

28 INDUSTRY-LEADING COMPANIES TO ENDORSE JAVASCRIPT AS A COMPLEMENT TO JAVA FOR EASY ONLINE APPLICATION DEVELOPMENT

MOUNTAIN VIEW, Calif. (December 4, 1995) -- Netscape Communications Corporation (NASDAQ: NSCP) and Sun Microsystems, Inc. (NASDAQ: SUNW), today announced JavaScript, an open, cross-platform object scripting language for the creation and customization of applications on enterprise networks and the Internet. The JavaScript language complements Java, Sun's industry-leading object-oriented, cross-platform programming language. The initial version of JavaScript is available now as part of the beta version of Netscape Navigator 2.0, which is currently available for downloading from Netscape's web site.

In addition, 28 industry-leading companies, including America Online, Inc., Apple Computer, Inc., Architext Software, Attachmate Corporation, AT&T, Borland International, Brio Technology, Inc., Computer Associates, Inc., Digital Equipment Corporation, Hewlett-Packard Company, Iconovox Corporation, Illustra Information Technologies, Inc., Informix Software, Inc., Intuit, Inc., Macromedia, Metrowerks, Inc., Novell, Inc., Oracle Corporation, Paper Software, Inc., Precept Software, Inc., RAD Technologies, Inc., The Santa Cruz Operation, Inc., Silicon Graphics, Inc., Spider Technologies, Sybase, Inc., Toshiba Corporation, Verity, Inc., and Vermeer Technologies, Inc., have endorsed JavaScript as an open standard object scripting language and intend to provide it in future products. The draft specification of JavaScript, as well as the final draft specification of Java, is planned for publishing and submission to appropriate standards bodies for industry review and comment this month.

JavaScript is an easy-to-use object scripting language designed for creating live online applications that link together objects and resources on both clients and servers. While Java is used by programmers to create new objects and applets, JavaScript is designed for use by HTML page authors and enterprise application developers to dynamically script the behavior of objects running on either the client or the server. JavaScript is analogous to Visual Basic in that it can be used by people with little or no programming experience to quickly construct complex applications. JavaScript's design represents the next generation of software designed specifically for the Internet and is:

- designed for creating network-centric applications
- complementary to and integrated with Java
- complementary to and integrated with HTML
- open and cross-platform

CARATTERISTICHE DI JAVASCRIPT - 2.5



LINGUAGGIO COMPILOTTO: UN PROGRAMMA (COMPILER) CONVERTE TUTTO IL CODICE IN LINGUAGGIO MACCHINA PRIMA CHE SIA ESEGUITO.

LINGUAGGIO INTERPRETATO: UN PROGRAMMA (INTERPRETER) CONVERTE ED ESEGUE IL CODICE RIGA PER RIGA.

PER LO STESSO LINGUAGGIO CI POSSONO ESSERE PIÙ COMPILER/INTERPRETER.

CARATTERISTICHE DI JAVASCRIPT - 2.6



- 1995: MICROSOFT RILASCIÀ «INTERNET EXPLORER» E CREA IL PROPRIO INTERPRETER JS CHIAMATO «JSCRIPT», NOTEVOLMENTE DIVERSO DA QUELLO DI NETSCAPE.
- 1996: NETSCAPE INVIA JAVASCRIPT ALL'ASSOCIAZIONE «ECMA INTERNATIONAL», CON L'OBIETTIVO DI DEFINIRE UNO STANDARD PER TUTTI I PRODUTTORI DI BROWSER.
- 1997 NASCE LO STANDARD ECMASCIPT 1.

CARATTERISTICHE DI JAVASCRIPT - 2.7



CARATTERISTICHE DI JAVASCRIPT - 2.8



- MICROSOFT RIFIUTA DI PARTECIPARE ALLE CONFERENZE ECMA (DAL 1999 AL 2009), QUINDI LA SITUAZIONE RESTA PIÙ O MENO LA STESSA FINO AL 2004, QUANDO MOZILLA RILASCIÀ IL BROWSER «FIREFOX»: È LA SCINTILLA DELLA RINASCITA DI JAVASCRIPT, INCENTRATA SULLE LIBRERIE OPEN-SOURCE E SULLE COMMUNITY ATTORNO AD ESSE.
- 2008: GOOGLE RILASCIÀ IL BROWSER «CHROME» CON IL JAVASCRIPT ENGINE «V8» CHE UTILIZZA PER LA PRIMA VOLTA LA COMPILAZIONE «JUST-IN-TIME» (JIT)

CARATTERISTICHE DI JAVASCRIPT - 2.9



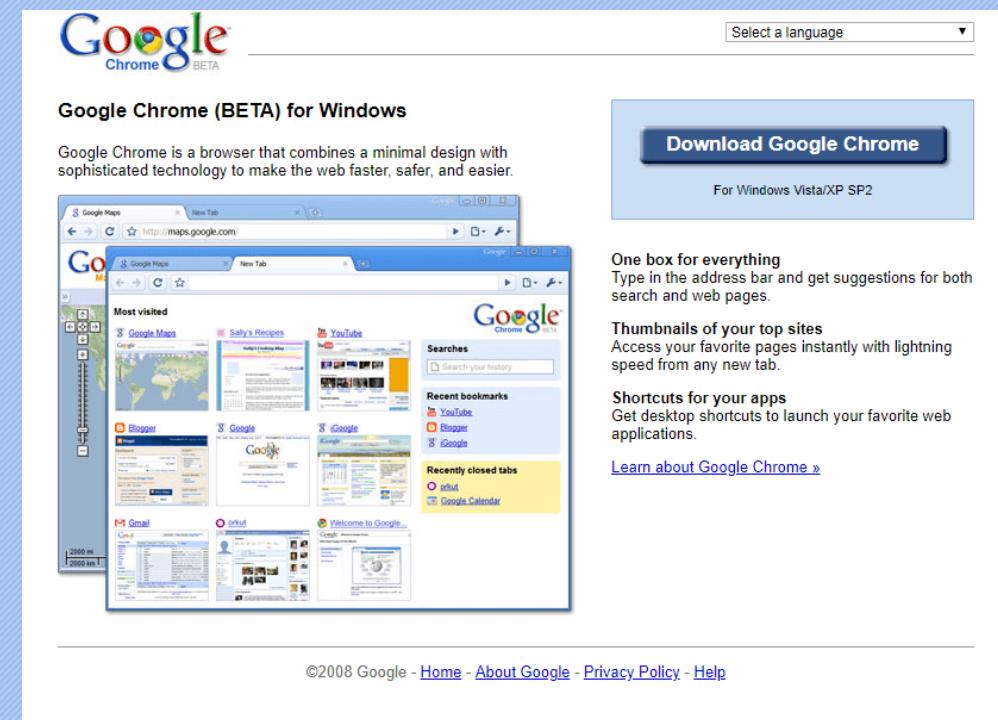
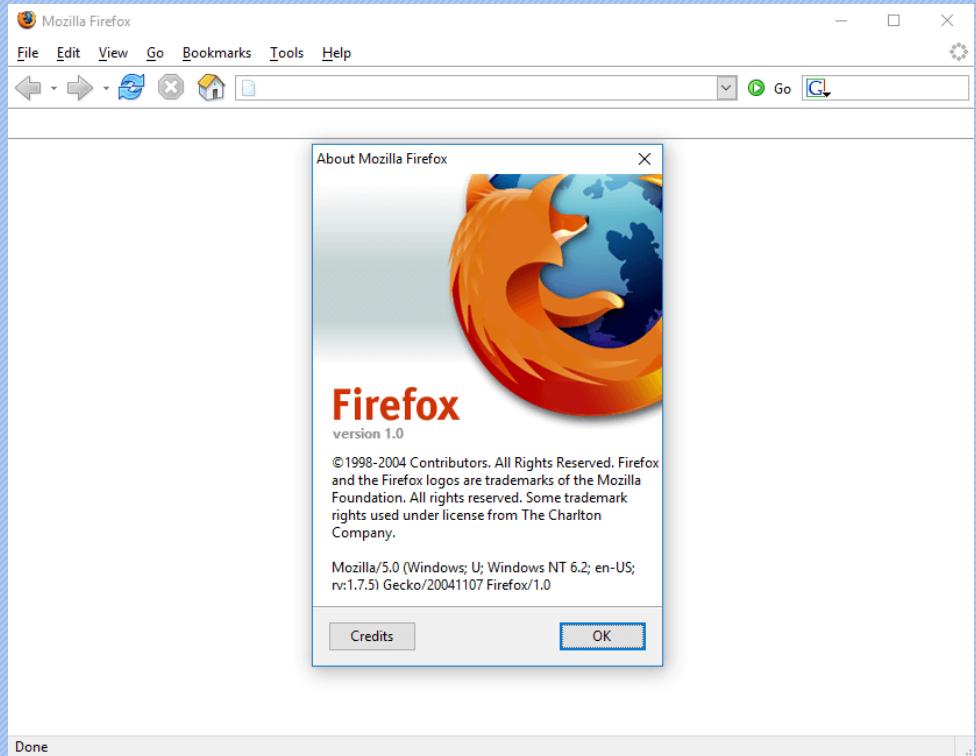
COMPILAZIONE «JUST-IN-TIME»:

IL CODICE VIENE CONVERTITO TRAMITE INTERPRETER NON IN LINGUAGGIO MACCHINA MA IN UN LINGUAGGIO INTERMEDIO DETTO «BYTECODE».

IL COMPILATORE INDIVIDUA LE PORZIONI DI «BYTECODE» ESEGUITE PIÙ SPESO E LE COMPILA IN CODICE MACCHINA: QUESTA COMPILAZIONE AVVIENE SUBITO PRIMA CHE IL CODICE SIA ESEGUITO (DA QUI IL NOME).

DOPO LA COMPILAZIONE, IL CODICE È MANTENUTO IN MEMORIA E RIUTILIZZATO SENZA ESSERE RICOMPILATO.

CARATTERISTICHE DI JAVASCRIPT - 2.10



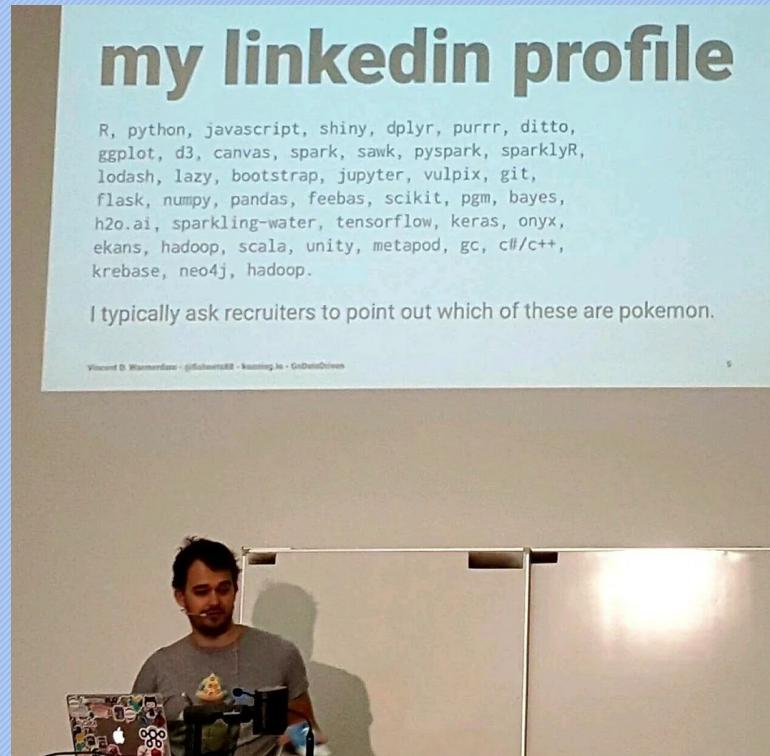
The screenshot shows the Google Chrome (BETA) for Windows landing page. It features the Google Chrome logo and the text 'Google Chrome (BETA) for Windows'. Below this, it says 'Google Chrome is a browser that combines a minimal design with sophisticated technology to make the web faster, safer, and easier.' A large screenshot of the browser interface is shown, highlighting features like 'Most visited' sites, 'Recent bookmarks', and 'Recently closed tabs'. To the right, there's a 'Download Google Chrome' button and a note for 'Windows Vista/XP SP2'. Below the download area, there are three sections: 'One box for everything' (describing the address bar), 'Thumbnails of your top sites' (describing the new tab page), and 'Shortcuts for your apps' (describing desktop integration). A link 'Learn about Google Chrome »' is also present.

CARATTERISTICHE DI JAVASCRIPT - 2.11

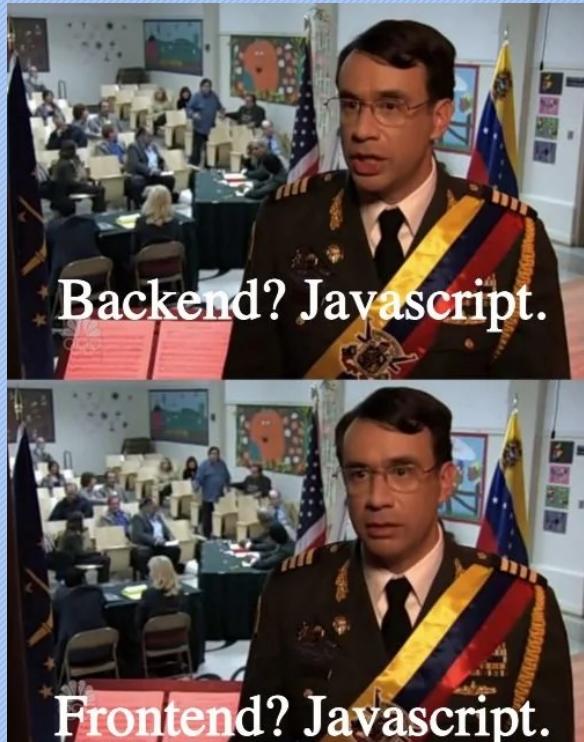


- 2009: CON LA CREAZIONE DI «NODE.JS», JAVASCRIPT PUÒ ESSERE USATO ANCHE LATO SERVER E NON SOLO ALL'INTERNO DI UN WEB BROWSER, RILANCIANDO L'IDEA DI UNIFICARE LO SVILUPPO DI APPLICATIVI INTORNO A UN SOLO LINGUAGGIO.
- ADESSO:
 - «STATE OF JS 2022»: <https://tsh.io/blog/javascript-trends/>
 - ECMASCIPT 14 (2023)
 - LIBRERIE E FRAMEWORKS (ANGULAR, REACT, VUE, EMBER, SVELTE, FLUTTER, LIT, STENCIL, NEXTJS, ASTRO, ELEVENTY, ELECTRON, IONIC, CORDOVA...)

CARATTERISTICHE DI JAVASCRIPT - 2.12



CARATTERISTICHE DI JAVASCRIPT - 2.13



Infrastructure? Believe it or not, also Javascript.

CARATTERISTICHE DI JAVASCRIPT - 3



JAVASCRIPT È UN LINGUAGGIO STANDARDIZZATO

«ECMA INTERNATIONAL» È UN ASSOCIAZIONE NO-PROFIT CON
L'OBBIETTIVO DI CREARE STANDARD NEI SETTORI INFORMATICO E
DELLA COMUNICAZIONE.

TRA I MEMBRI CI SONO APPLE, IBM, GOOGLE, META, HUAWEI...

IL DOCUMENTO ECMA-262 (ANCHE «ECMASCRIPT» O «ES») DEFINISCE
LO STANDARD A CUI TUTTI I PRODUTTORI DOVREBBERO CONFORMARE
I PROPRI JAVASCRIPT ENGINE.

CARATTERISTICHE DI JAVASCRIPT - 3.1



NELLA PRATICA, I BROWSER NON SEGUONO LE SPECIFICHE AL 100%:

ENGINE	BROWSER	ES5 (2009)	ES6 (2015)	ES2016+	NEXT
SPIDERMONKEY	FIREFOX 120	100%	98%	98%	5%
V8	CHROME 117 EDGE 113	100%	98%	98%	5%
JAVASCRIPTCODE	SAFARI 17	99%	100%	98%	11%

<https://compat-table.github.io/compat-table/esnext/>

CARATTERISTICHE DI JAVASCRIPT - 3.2



COSA FA PARTE DELLO STANDARD?

- LA SINTASSI DEL LINGUAGGIO (KEYWORDS, CONTROL FLOW...);
- LA GESTIONE DEGLI ERRORI (THROW, TRY..CATCH, CUSTOM ERROR...);
- I TIPI STANDARD (BOOLEAN, NUMBER, STRING, FUNCTION, OBJECT...);
- L'EREDITARIETÀ PROTOTYPE-BASED;
- GLI OGGETTI E LE FUNZIONI INTEGRATE NEL LINGUAGGIO (JSON, MATH...);
- ...

LA COMMISSIONE SI RADUNA OGNI DUE MESI PER DISCUTERE SU NUOVE PROPOSTE DELLA COMMUNITY (NUOVE FEATURES, MIGLIORARE LA DOCUMENTAZIONE, AGGIUNGERE TEST...).

CARATTERISTICHE DI JAVASCRIPT - 4



JAVASCRIPT È UN LINGUAGGIO IMPERATIVO E STRUTTURATO.

IMPERATIVO: UNO SCRIPT JS CONSISTE IN UN INSIEME DI ISTRUZIONI, OVVERO COMANDI, CHE LA MACCHINA ESEGUE PER CAMBIARE LO STATO DEL SISTEMA.

STRUTTURATO: LE ISTRUZIONI SONO ESEGUITE UTILIZZANDO

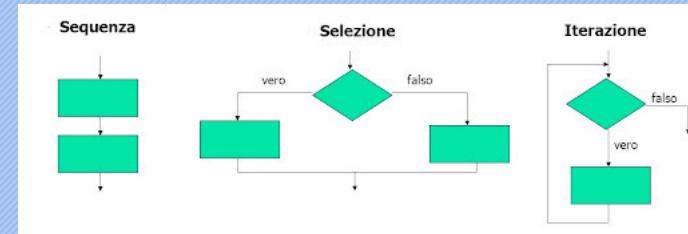
- I COSTRUTTI TIPICI DEI DIAGRAMMI DI FLUSSO (SEQUENZA, SELEZIONE, ITERAZIONE);
- SUBROUTINE O «SOTTOPROGRAMMA» (PROCEDURE, FUNZIONI, METODI..);
- BLOCCHI.

ESISTONO ECCEZIONI: EARLY EXIT, EXCEPTION HANDLING..

CARATTERISTICHE DI JAVASCRIPT - 4.1

```
1 // sequence
2 alert('Hello');
3 alert('world!');
4
5 // selection
6 if (true)
7   alert('Hello world!');
8
9 // iteration
10 do
11   alert('Hello world!')
12 while (true);
13
14 // subroutine and block
15 const myAlert = () :void => {
16   alert('Hello world!');
17 }
18 myAlert();
19
```

TEOREMA DI BÖHM - JACOPINI (1966)



CARATTERISTICHE DI JAVASCRIPT - 5



JAVASCRIPT È UN LINGUAGGIO DEBOLMENTE TIPIZZATO:

- È POSSIBILE ASSEGNAME VARIABILI SENZA DEFINIRNE IL TIPO
- A SECONDA DELL'OPERATORE USATO, SI PUÒ AVERE UN CASTING IMPLICITO
- EVENTUALI TYPE ERRORS APPAIONO A RUNTIME
- SI PUÒ RINFORZARE IL CONTROLLO CON «use strict»

TIPI PRIMITIVI:

- number
- bigint
- string
- boolean
- null
- undefined
- symbol

TIPO NON PRIMITIVO:

- object

CARATTERISTICHE DI JAVASCRIPT - 5.1

```
> typeof NaN           > true==1          > Math.min()          > []==0
< "number"            < true             < Infinity          < true
> 9999999999999999  > true==1          > []+[]
< 10000000000000000  < false            < ""
> 0.5+0.1==0.6       > (!+[]+[]+![]).length
< true                < 9                 > []+{}
< 0.1+0.2==0.3       > 9+"1"            < "[object Object]"
< false               < "91"              > {}+[]
> Math.max()          > 91-1"            < 0
< -Infinity          < 90                > true+true+true==3
<
```



CARATTERISTICHE DI JAVASCRIPT - 5.2



JAVASCRIPT È UN LINGUAGGIO DINAMICO.

DURANTE L'ESECUZIONE DEL CODICE

- È POSSIBILE CAMBIARE IL TIPO DELLE VARIABILI
- È POSSIBILE CAMBIARE IL VALORE DELLE VARIABILI
- È POSSIBILE AGGIUNGERE NUOVO CODICE, ESTENDERE CLASSI E CREARE NUOVE DEFINIZIONI
- È POSSIBILE «ISPEZIONARE» IL CODICE
- ...

CARATTERISTICHE DI JAVASCRIPT - 6



JAVASCRIPT È UN LINGUAGGIO ORIENTATO AGLI OGGETTI.

UN OGGETTO È UNA COLLEZIONE DI PROPRIETÀ; UNA PROPRIETÀ È UN'ASSOCIAZIONE TRA UN NOME («CHIAVE») E UN VALORE.

IL VALORE DI UNA PROPRIETÀ PUÒ ESSERE DI QUALUNQUE TIPO
(ANCHE UN ALTRO OGGETTO);
SE È UNA FUNZIONE, ALLORA LA PROPRIETÀ È DETTA «METODO».

L'EREDITARIETÀ TRA OGGETTI FUNZIONA TRAMITE PROTOTIPO.

CARATTERISTICHE DI JAVASCRIPT - 6.1

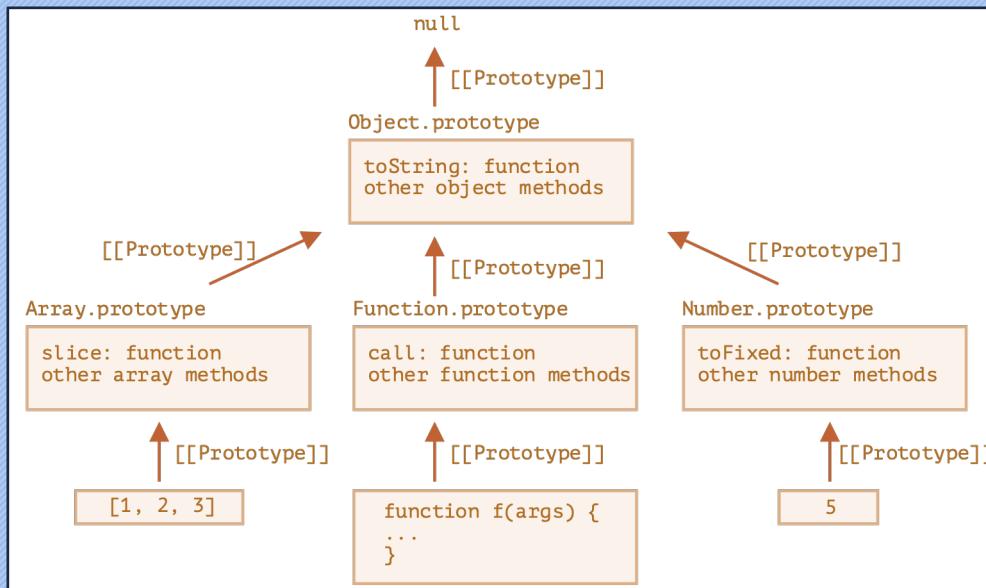


IL PROTOTIPO DI UN OGGETTO È UN ALTRO OGGETTO A CUI IL PRIMO È COLLEGATO; A SUA VOLTA ESSO AVRÀ UN SUO PROTOTIPO E COSÌ VIA («PROTOTYPE CHAIN»). IL PROTOTIPO BASE È «null».

OGNI OGGETTO EREDITA LE PROPRIETÀ/METODI DEGLI OGGETTI NELLA SUA «PROTOTYPE CHAIN»; INOLTRE, È POSSIBILE MODIFICARE DINAMICAMENTE OGNI ELEMENTO NELLA «PROTOTYPE CHAIN».

IL MODO PIÙ COMUNE DI DEFINIRE IL PROTOTIPO DI UN OGGETTO È TRAMITE COSTRUTTORE. IL CONCETTO DI CLASSE È UN'ASTRAZIONE ULTERIORE BASATA APPUNTO SU PROTOTIPO E COSTRUTTORE.

CARATTERISTICHE DI JAVASCRIPT - 6.2



```
1 const personPrototype :{} = {  
2     introduceSelf: function () :void {  
3         alert(`Hi! I'm ${this.name.firstName}.`);  
4     };  
5 };  
6  
7 const person :{} = {  
8     name: {  
9         firstName: 'Davide',  
10        lastName: 'Mininni',  
11    },  
12    age: 34,  
13    __proto__: personPrototype,  
14 };  
15  
16 person.introduceSelf();  
17 // Hi! I'm Davide.
```

CARATTERISTICHE DI JAVASCRIPT - 6.3

```
1 function Person(firstName, lastName, age) :void { Show usages
2     this.name = {
3         firstName: firstName,
4         lastName: lastName
5     };
6     this.age = age;
7 }
8
9 Person.prototype.introduceSelf = function () :void {
10     alert(`Hi! I'm ${this.name.firstName}.`);
11 }
12
13 const person :Person = new Person( firstName: 'Davide', lastName: 'Mininni', age: 34);
14 person.introduceSelf();
15 // Hi! I'm Davide.
```

```
1 class Person { Show usages
2     constructor(firstName, lastName, age) { Show usages
3         this.name = {
4             firstName: firstName,
5             lastName: lastName
6         };
7         this.age = age;
8     }
9     introduceSelf() :void { Show usages
10     alert(`Hi! I'm ${this.name.firstName}.`);
11 }
12
13 const person :Person = new Person( firstName: 'Davide', lastName: 'Mininni', age: 34);
14 person.introduceSelf();
15 // Hi! I'm Davide.
```

CARATTERISTICHE DI JAVASCRIPT - 7



JAVASCRIPT È UN LINGUAGGIO FUNZIONALE.

LE FUNZIONI IN JAVASCRIPT SONO «FIRST-CLASS CITIZENS»:
POSSENO ESSERE ASSEGNAI A VARIABILI O A PROPRIETÀ, PASSATE
COME ARGOMENTO DI ALTRE FUNZIONI O ESSERE VALORE DI USCITA DI
ALTRE FUNZIONI.

ALL'INTERNO DI UNA FUNZIONE POSSONO ANCHE ESSERE DEFINITE
PROPRIETÀ O ALTRE FUNZIONI.

CARATTERISTICHE DI JAVASCRIPT - 7.1

```
1  const numList :number[] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
2  let result :number = 0;
3  for (let i :number = 0; i < numList.length; i++) {
4      if (numList[i] % 2 === 0) {
5          result += numList[i] * 10;
6      }
7  }
8
9  const resultFunc = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
10     .filter(n :number => n % 2 === 0) number[]
11     .map(a :number => a * 10) unknown[]
12     .reduce((a, b) => a + b, 0);
```

JAVASCRIPT NEL BROWSER - 1

CI CONCENTREREMO SULL'UTILIZZO DI JAVASCRIPT NEL BROWSER.

IL TAG ‘`<script>`’ PUÒ ESSERE USATO PER AGGIUNGERE CODICE JAVASCRIPT A UNA PAGINA HTML.

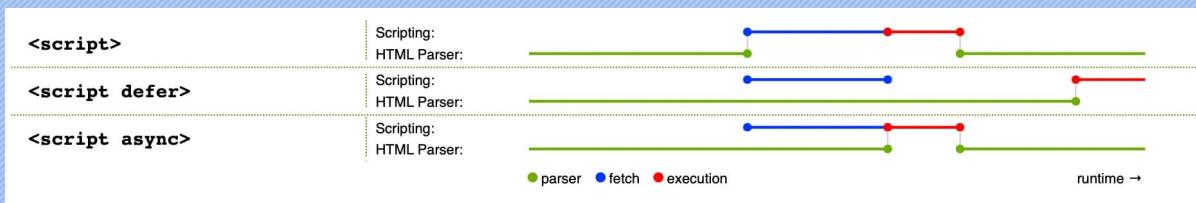
IN ALTERNATIVA SI PUÒ CREARE UN FILE CON ESTENSIONE ‘.js’ E INCLUDERLO NELL’HTML USANDO L’ATTRIBUTO ‘src’ DEL TAG ‘`<script>`’.

```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8">
5          <title>Demo</title>
6          <link rel="stylesheet" href=".style.css">
7          <script src=".script.js" defer></script>
8      </head>
9      <body>
10         <script>
11             alert('Hello world');
12         </script>
13     </body>
14 </html>
15
```

JAVASCRIPT NEL BROWSER - 1.1

IL TAG ‘`<script>`’ HA ALTRI ATTRIBUTI, ALCUNI DEI QUALI CADUTI IN DISUSO:

- ‘type’ ERA RICHIESTO IN HTML 4, ORA NON PIÙ; TUTTAVIA PUÒ ESSERE USATO PER CARicare MODULI JAVASCRIPT
- ‘defer’ FA SI CHE IL BROWSER CONTINUI IL PARSING DELL’HTML MENTRE LO SCRIPT VIENE SCARICATO; L’ESECUZIONE È POSTICIPATA AL COMPLETAMENTO DEL PARSING
- CON ‘async’ INVECE, LO SCRIPT VIENE ESEGUITO SUBITO DOPO IL DOWNLOAD, POTENZIALMENTE INTERROMPENDO IL PARSING



JAVASCRIPT NEL BROWSER - 1.2

È BUONA NORMA NON INSERIRE CODICE JAVASCRIPT NELL'HTML.

CARICANDO UN FILE ESTERNO:

- OGNI FILE HA UN'UNICA FUNZIONE;
- IL CODICE È PIÙ FACILE DA LEGGERE E MANTENERE;
- IL BROWSER MEMORIZZA IL FILE IN CACHE.

È POSSIBILE CARICARE INTERE LIBRERIE (ES. TRAMITE LINK A CDN).



```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6      <link rel="stylesheet" href=".style.css">
7      <script src=".script.js" defer></script>
8    </head>
9    <body>
10   <script>
11     alert('Hello world');
12   </script>
13 </body>
14 </html>
15
```

JAVASCRIPT NEL BROWSER - 1.3



ESERCIZIO:

- CREARE UN NUOVO PROGETTO CON TRE FILES (HTML, CSS, JS);
- CARICARE I DUE FILE ESTERNI NELL'HTML;
- INSERIRE UN ALERT CON UN MESSAGGIO NEL JS;
- VISUALIZZARE L'ALERT.

TEMPO: 10 MINUTI

DEBUGGING - 1

COME VISUALIZZARE O DEBUGGARE UN OUTPUT?

- WINDOW API:
alert / prompt / confirm
- CONSOLE API:
 - console.log/warn/error
 - console.time/timeLog/timeEnd
 - console.table
 - console.trace
- DOCUMENT: innerHtml

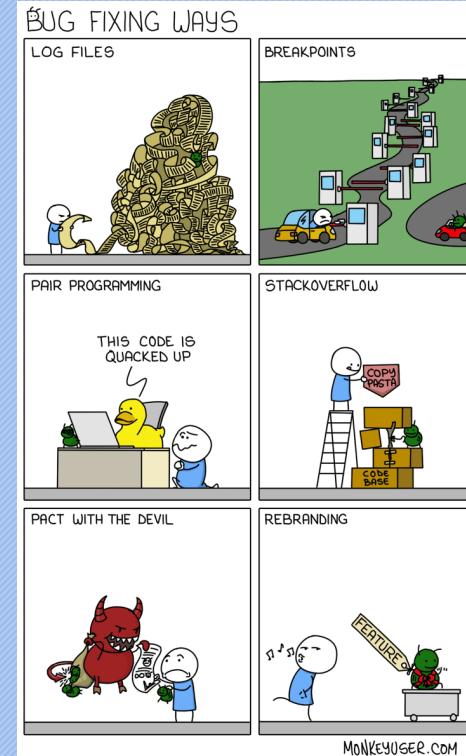
```
> console.log('Hello world!')
Hello world!
< undefined
> console.warn('Hello world!')
⚠ ▶ Hello world!
< undefined
> console.error('Hello world!')
✖ ▶ Hello world!
< undefined
> console.time();
console.log('Hello world!');
console.timeLog();
console.log('Hello world!');
console.timeEnd();
Hello world!
default: 0.172119140625 ms
Hello world!
default: 0.2841796875 ms
< undefined
```

DEBUGGING - 1.1

ESERCIZIO:

- USANDO «prompt» APRIRE UNA DIALOG CHE RICHIEDE UN NOME;
- VISUALIZZARE IL NOME USANDO «alert»;
- LOGGARE IL NOME IN CONSOLE USANDO UNA DELLE API.

TEMPO: 10 MINUTI



STATEMENTS - 1



UNO SCRIPT JS PUÒ ESSERE VISTO COME UNA SERIE DI «ISTRUZIONI» CHE IL BROWSER «ESEGUE».

LE ISTRUZIONI SONO CHIAMATE «STATEMENTS».

OGNI «STATEMENT» SI CHIUDE CON UN PUNTO E VIRGOLA; È BUONA NORMA AGGIUNGERE SPAZI TRA LE PAROLE PER MIGLIORARE LA LEGGIBILITÀ, E COMMENTI PER SPIEGARE COSA SI STA FACENDO.

GLI «STATEMENTS» POSSONO ESSERE RAGGRUPPATI IN BLOCCHI DI CODICE USANDO LE PARENTESI GRAFFE.

VARIABILI - 1



PER MEMORIZZARE DATI SI UTILIZZANO LE VARIABILI, DICHIARANDOLE CON LE KEYWORDS
«var» - «let» - «const».

IL SIMBOLO «=» VIENE USATO PER ASSEGNARE ALLA VARIABILE IL VALORE DA MEMORIZZARE. FINCHÉ NON È ASSEGNATA, VALE «undefined».

```
1  let a, b, c;           // Declare 3 variables
2  a = 5;                // Assign the value 5 to a
3  b = 6;                // Assign the value 6 to b
4  c = a + b;             // Assign the sum of a and b to c
5  console.log(c);        // Displays the sum value in the console.
```

VARIABILI - 2



ESISTONO REGOLE PER I NOMI DELLE VARIABILI:

- SI USA IL MINUSCOLO - OCCHIO AL CASING;
- SOLO LETTERE E NUMERI, NO CARATTERI SPECIALI TRANNE «\$» / «_»;
- NON SI INIZIA CON UN NUMERO;
- NOMI COMPOSTI IN «CAMEL CASE»;
- LE LETTERE NON LATINE SONO PERMESSE, MA NON RACCOMANDATE.

PER LE COSTANTI VALGONO LE STESSE REGOLE, MA USANDO LO «SCREAMING SNAKE CASE».

ALCUNE PAROLE SONO PROIBITE PERCHÉ USATE DAL LINGUAGGIO STESSO.

VARIABILI - 3



VARIABILI - 4

KEYWORD	DESCRIZIONE
var	DICHIARA UNA VARIABILE GLOBALE (OLD GEN BROWSERS)
let	DICHIARA UNA VARIABILE LOCALE
const	DICHIARA UNA COSTANTE LOCALE
if /else if / else	IL CODICE VIENE ESEGUITO SE LA CONDIZIONE È VERA, ALTRIMENTI SI PASSA ALLA CONDIZIONE SUCCESSIVA
switch	IL CODICE VIENE ESEGUITO A SECONDA DELLE CONDIZIONI
for	IL CODICE VIENE ESEGUITO IN LOOP
function	DICHIARA UNA FUNZIONE
return	ESCE DA UNA FUNZIONE

```
1  let minutesInADay : number = 1440;
2
3  const MINUTES_IN_HOUR : number = 60;
4  const HOURS_IN_DAY : number = 24;
5
6  function _convertMinutesInHours(minutes) { Show usages
7      return minutes / MINUTES_IN_HOUR;
8  }
9
10 function convertHoursInDays(hours) { Show usages
11     return hours / HOURS_IN_DAY;
12 }
13
14 function convertMinutesInDays(minutes) { Show usages
15     return convertHoursInDays(_convertMinutesInHours(minutes));
16 }
17
18 console.assert( condition: convertMinutesInDays(minutesInADay) === 1);
```

DATA TYPES - 1



ESISTONO 8 TIPI IN JAVASCRIPT, DI CUI 7 PRIMITIVI E 1 NON PRIMITIVO (VEDI SLIDES PRECEDENTI).

NE VEDREMO ORA QUALCHE DETTAGLIO GENERALE.

UNA VARIABILE PUÒ ESSERE ASSEGNATA A UN VALORE DI QUALUNQUE TIPO;
POICHÉ JAVASCRIPT È «DINAMICAMENTE TIPIZZATO»,
LA VARIABILE PUÒ ESSERE RIASSEGNATA A UN VALORE DI TIPO DIVERSO SENZA GENERARE ERRORI.

DATA TYPES - 2

IL TIPO «number» PUÒ RAPPRESENTARE NUMERI SIA INTERI CHE DECIMALI.

PER OPERARE SU VARIABILI NUMERICHE SI POSSONO USARE GLI OPERATORI ALGEBRICI + - * / OLTRE AGLI OPERATORI % **

«Infinity», «-Infinity» E «NaN» SONO VALORI NUMERICI SPECIALI.

```
1 const n1 : number = 1;
2 const n2 : number = 12345.6789;
3 const n3 : number = 1_000_000_000;

4
5 console.log( n1 / 0 );
// Infinity

7
8 console.log( 'A string' * n2 );
// NaN

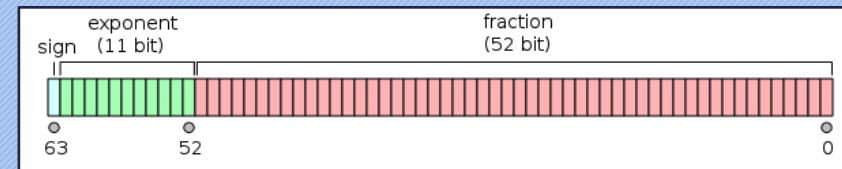
10
11 console.log( NaN + n3 );
// NaN
```



DATA TYPES - 3

IL TIPO «number» È FORMATTATO COME «DOUBLE-PRECISION 64-BIT FLOATING-POINT» E RAPPRESENTA CORRETTAMENTE I NUMERI SOLO NEL RANGE $\pm(2^{53}-1)$.

PER OPERAZIONI DI PRECISIONE SI PUÒ USARE IL TIPO «BigInt», CHE INVECE SUPPORTA INTERI DI LUNGHEZZA ARBITRARIA.



$$(-1)^{\text{sign}} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i} \right) \times 2^{e-1023}$$

DATA TYPES - 3.1



SI CREA UN «`BigInt`» APPENDENDO UNA «`n`» ALLA FINE DI UN NUMERO O USANDO LA FUNZIONE «`BigInt(...)`».

NELLE OPERAZIONI, NON SI POSSONO MISCHIARE «`number`» E «`BigInt`». GLI OPERATORI FUNZIONANO TUTTI, ESCLUSO `>>>`.

```
1  const n1 : number = 1;
2  const n2 : number = 12345.6789;
3  const n3 : number = 1_000_000_000;
4
5  const big1 : bigint = 1n;
6  const big2 : bigint = 1_000_000_000n;
7  const big3 : bigint = BigInt( value: 10000000000000 );
8  const big4 : bigint = BigInt( value: '10000000000000' );
9
10 console.assert( condition: 0n === 0 );
11 // false
12 console.assert( condition: 0n == 0 );
13 // true
```

DATA TYPES - 4

UNA «string» È UN DATO DI TESTO;
VIENE SEMPRE RACCHIUSA DA APICI

- SINGOLI
- DOPPI
- BACKTICKS

IL BACKTICK PERMETTE L'USO DI
ESPRESSIONI TRAMITE «\${...}»

NON ESISTE UN TIPO PER IL
SINGOLO CARATTERE
(«CHAR» IN C/JAVA).

```
1  const apex :string = 'ONE';
2  const double :string = "two";
3  const backtick :string = `This is a string with \`backticks\``;
4
5  const evalFunct :string = `The sum of 2 and 2 is ${2 + 2}.`;
6  console.log(evalFunct);
7  // The sum of 2 and 2 is 4.
8
9  const lines :string = `I can
10   split text
11   in different lines
12   without error`;
13  const single :string = "I can\nsplit text\nin different lines\nwithout error";
14  console.assert( condition: single === lines);
15  // true
16
17  const string1 :string = 'String can be concatenated ';
18  const string2 :string = 'using the';
19  console.log(string1 + string2 + ' + operator.');
20  // String can be concatenated using the + operator.
```



DATA TYPES - 5

IL TIPO «boolean» AMMETTE SOLO DUE VALORI: «true» E «false».

VALORI BOOLEANI SONO COMUNEMENTE UTILIZZATI CON GLI OPERATORI LOGICI E SONO RESTITUITI DAGLI OPERATORI DI CONFRONTO; SONO OVVIAIMENTE USATI ANCHE PER IL «CONTROL FLOW» DELLO SCRIPT.

```
1  const trueValue :boolean = true;
2  const falseValue :boolean = false;
3
4  const trueOr :boolean = falseValue || trueValue;
5  console.log(trueOr);
6  // true
7
8  const falseAnd :boolean = falseValue && trueValue;
9  console.log(falseAnd);
10 // false
11
12 if (trueValue) {
13   console.log('if branch')
14 } else {
15   console.log('else branch')
16 }
17 // if branch
```

DATA TYPES - 6



I TIPI «null» E «undefined» SONO SPECIALI E A SE STANTI.
«null» RAPPRESENTA «VUOTO», «VALORE SCONOSCIUTO»;
«undefined» RAPPRESENTA «VALORE NON (ANCORA) ASSEGNATO».

IL TIPO «object» È NON PRIMITIVO; ABBIAMO VISTO CHE UN OGGETTO È UNA COLLEZIONE DI PROPRIETÀ (CHIAVE/VALORE). GLI OGGETTI SONO USATI PER CONTENERE DATI ED ENTITÀ COMPLESSE. ANCHE IL TIPO «symbol» È SPECIALE; È USATO PER CREARE CHIAVI UNIVOCHE PER GLI OGGETTI. VEDREMO ALTRI DETTAGLI IN SEGUITO.

CONVERSIONE - 1

SEBBENE OPERATORI E FUNZIONI APPLICHINO AUTOMATICAMENTE UNA CONVERSIONE DEI DATI AL TIPO CORRETTO, A VOLTE È NECESSARIO UNA CONVERSIONE ESPLICITA.

PER «string», «number» E «boolean» SI UTILIZZA IL CORRISPETTIVO «OBJECT WRAPPER».



CONVERSIONE - 2



PER OGNI PRIMITIVA, TRANNE «null» E «undefined», ESISTE UN CORRISPETTIVO «OBJECT WRAPPER» CHE ESPONE METODI E PROPRIETÀ UTILI, E CHE PERMETTE DI FORZARE LA CONVERSIONE TRA TIPI DIVERSI.

GLI «OBJECT WRAPPER» SONO:
String, Number, BigInt, Boolean,
Symbol.

```
1  const number1 : number = 1000;
2  console.log(typeof number1);
3  // number
4  console.log(typeof String(number1), String(number1));
5  // string "1000"
6
7  const booleanToString : boolean = false;
8  console.log(typeof boolean1);
9  // boolean
10 console.log(typeof String(booleanToString), String(booleanToString));
11 // string "false"
```

CONVERSIONE - 3

```
1  console.log(Boolean( value: 1));
2  // true
3  console.log(Boolean( value: '0'));
4  // true
5  console.log(Boolean( value: "Hello world!"));
6  // true
7  console.log(Boolean( value: " "));
8  // true
9
10 console.log(Boolean( value: 0));
11 // false
12 console.log(Boolean( value: ""));
13 // false
14 console.log(Boolean( value: null));
15 // false
16 console.log(Boolean( value: undefined));
17 // false
18 console.log(Boolean(NaN));
19 // false
```

```
1  const string1 :string = '1';
2  console.log(typeof string1);
3  // string
4  console.log(typeof Number(string1), Number(string1));
5  // number 1
6
7  console.log(Number( value: undefined));
8  // NaN
9  console.log(Number( value: null));
10 // 0
11 console.log(Number( value: true));
12 // 1
13 console.log(Number( value: false));
14 // 0
15 console.log(Number( value: ''));
16 // 0
17 console.log(Number( value: ' 111 '));
18 // 111
19 console.log(Number( value: 'abcd'));
20 // NaN
```

OPERATORI - 1



ABBIAMO GIÀ INTRODOTTO L'OPERATORE «=»,
CHE VIENE USATO PER ASSEGNARE UN VALORE A UNA VARIABILE.

ESISTONO ALTRI OPERATORI:

- OPERATORI ALGEBRICI
- OPERATORI DI ASSEGNAZIONE
- OPERATORI DI CONFRONTO
- OPERATORI LOGICI
- «TYPE OPERATORS»
- «BITWISE OPERATORS»
- ...

OPERATORI - 2

OPERATORI DI CONFRONTO	
==	UGUALE
===	UGUALE (VALORE E TIPO)
!=	DIVERSO
!==	DIVERSO (VALORE E TIPO)
>	MAGGIORE DI
<	MINORE DI
>=	MAGGIORE O UGUALE A
<=	MINORE O UGUALE A
?	OPERATORE TERNARIO

OPERATORI ALGEBRICI	
+	ADDIZIONE
-	SOTTRAZIONE
*	MOLTIPLICAZIONE
**	ESPOLENTE (ES2016)
/	DIVISIONE
%	MODULO
++	INCREMENTO (PRE/POST)
--	DECREMENTO (PRE/POST)

OP. DI ASSEGNAZAMENTO		
=	x = y	x = y
+=	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
**=	x **= y	x = x ** y
[a, b] = arr, { a, b } = obj		

OPERATORI - 3

OPERATORI LOGICI	
&&	«AND» LOGICO
	«OR» LOGICO
!	«NOT» LOGICO
??	«NULLISH COALESCING»
TYPE OPERATORS	
typeof	RESTITUISCE IL TIPO DELLA VARIABILE
instanceof	RESTITUISCE true SE NELLA «PROTOTYPE CHAIN» DELL'OGGETTO C'È IL TIPO RICHIESTO
BITWISE OPERATORS	
&	AND
	OR
~	NOT
^	XOR
<<	LEFT SHIFT
>>	RIGHT SHIFT
>>>	UNSIGNED RIGHT SHIFT

OPERATORI - 4



TERMINOLOGIA:

- OPERANDO: A COSA APPLICO L'OPERATORE;
- OPERATORE UNARIO: SE SI APPLICA A UN SOLO OPERANDO;
- OPERATORE BINARIO: SE SI APPLICA A UNA COPPIA DI OPERATORI.

OPERATORI ALGEBRICI:

- QUATTRO OPERAZIONI DI BASE: «+» «-» «*» «/»
- «%» : RESTITUISCE IL RESTO DELLA DIVISIONE
- «**» : RESTITUISCE IL PRIMO OPERANDO ELEVATO AL SECONDO
- «++» «--»: INCREMENTA/DECREMENTA LA VARIABILE DI 1. IL VALORE RITORNATO DIPENDE DALLA POSIZIONE (PREFIX: NUOVO, POSTFIX: VECCHIO)

OPERATORI - 5



OPERATORI DI CONFRONTO:

- RESTITUISCONO UN «boolean»;
- «==» E «!=» SONO PREFERIBILI RISPETTO A «==>» E «!=>»;
- IL CONFRONTO TRA «string» VIENE EFFETTUATO POSIZIONALMENTE PER CARATTERE IN RIFERIMENTO ALLA TABELLA UNICODE;
- CONFRONTANDO «string» E «number», LE «string» SONO CONVERTITE;
- L'OPERATORE TERNARIO «?» EQUIVALE A UN «if... else».

OPERATORI LOGICI

- IL RISULTATO RISPECCHIA LE TAVOLE DI VERITÀ;
- «||» SI FERMA AL PRIMO VALORE true, «&&» SI FERMA AL PRIMO VALORE false;
- «&&» HA PRECEDENZA SU «||».

OPERATORI - 6



```
1  let a : number = 1, b : number = 1;
2  let c : number = ++a;
3  let d : number = b++;
4
5  console.log(a, b, c, d);
6  // what do you expect?
7
8
9  let x : number = 2;
10 let y : number = 1 + (x *= 2);
11
12 console.log(x, y);
13 // what do you expect?
```

OPERATORI - 7



```
1  console.log(  
2    "" + 1 + 0,  
3    "" - 1 + 0,  
4    true + false,  
5    6 / "3",  
6    "2" * "3",  
7    4 + 5 + "px",  
8    "$" + 4 + 5,  
9    "4" - 2,  
10   "4px" - 2,  
11   " -9 " + 5,  
12   " -9 " - 5,  
13   null + 1,  
14   undefined + 1,  
15   " \t \n" - 2,  
16 );  
17 // what do you expect?
```

CONTROL FLOW - 1

IN TERMINI DI «CONTROL FLOW» SI PARLA DI «SCELTA» QUANDO SI IL CODICE VIENE ESEGUITO O MENO IN BASE AL VERIFICARSI DI UNA CONDIZIONE BOOLEANA.

IN JAVASCRIPT SI UTILIZZANO LE KEYWORDS «if...», «... else if ...» E «else...».



CONTROL FLOW - 1.1



«if...»: ESEGUE IL CODICE SE LA CONDIZIONE È VERA

«if... else...»: ESEGUE IL CODICE NEL PRIMO BLOCCO SE LA CONDIZIONE È VERA, ALTRIMENTI ESEGUE IL CODICE DEL SECONDO BLOCCO

«..else if...»: AGGIUNGE ALTRI BLOCCHI CON ALTRE CONDIZIONI

```
1  const truthyValue :boolean = true;
2  const falsyValue :boolean = false;
3
4  if (truthyValue) {
5      console.log('Good morning!')
6  }
7  // Good morning!
8
9  if (falsyValue) {
10     console.log('Good morning!')
11 } else {
12     console.log('Good night!')
13 }
14 // Good night!
15 falsyValue ? console.log('Good morning!') : console.log('Good night!');
16 // Good night!
17
18 if (truthyValue && falsyValue) {
19     console.log('Good morning!')
20 } else if (truthyValue) {
21     console.log('Good afternoon!')
22 } else {
23     console.log('Good night!')
24 }
25 // Good afternoon
```

CONTROL FLOW - 1.2

QUANDO SI VUOLE CONFRONTARE IL VALORE DI UNA VARIABILE RISPETTO A UN SET DI VALORI INVECE DEI COSTRUTTI PRECEDENTI SI PUÒ USARE IL COSTRUTTO «switch».

LA KEYWORD «break» SI USA PER INTERROMPERE L'ESECUZIONE; LA KEYWORD «default» SI USA COME CASO DI FALBACK.

```
1  let fruit :string = prompt( message: 'Fruit prices:');
2
3  switch (fruit) {
4    case "Oranges":
5      alert("Oranges are $0.59 a pound.");
6      break;
7    case "Apples":
8      alert("Apples are $0.32 a pound.");
9      break;
10   case "Bananas":
11     alert("Bananas are $0.48 a pound.");
12     break;
13   case "Cherries":
14     alert("Cherries are $3.00 a pound.");
15     break;
16   case "Mangoes":
17   case "Papayas":
18     alert("Mangoes and papayas are $2.79 a pound.");
19     break;
20   default:
21     alert(`Sorry, we are out of ${expr}.`);
22 }
```



CONTROL FLOW - 1.3

I SINGOLI CASE POSSONO ESSERE ANCHE RAGGRUPPATI.

SENZA IL «break» IL CODICE CONTINUA AL CASE SUCCESSIVO SENZA ALCUN CONTROLLO.

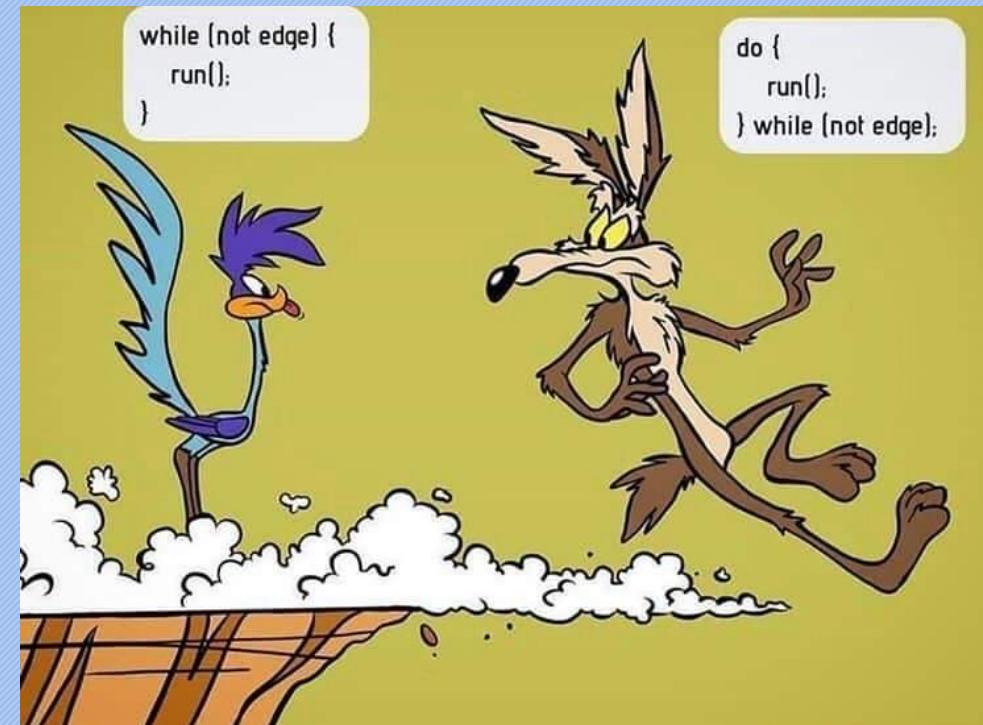
SE SI DEVONO ESEGUIRE PIÙ RIGHE, SI INSERISCE UN BLOCCO USANDO LE GRAFFE.

```
1  let four : number = Number(prompt( message: '2 + 2 = ?' , _default: 4));
2
3  switch (four) {
4    case 0:
5    case 1:
6    case 2:
7      console.log('Lower!');
8    case 3: {
9      console.log('Lower!');
10     break;
11   }
12   case 4:
13     console.log('Exact!');
14     break;
15   default:
16     console.log('Higher!');
17 }
```

CONTROL FLOW - 2

ANCHE I «LOOPS» FANNO PARTE DELLE ISTRUZIONI DI «CONDITIONAL BRANCHING»: UN LOOP È UN BLOCCO DI CODICE ESEGUITO PIÙ VOLTE IN BASE ALLE CONDIZIONI.

IN JAVASCRIPT SI UTILIZZANO LE KEYWORDS «while...», «do...while...» E «for...».



CONTROL FLOW - 2.1

IL «while» CONTROLLA SE LA CONDIZIONE È VERA, QUINDI ESEGUE IL CODICE NEL «LOOP BODY». UNA SINGOLA ESECUZIONE È CHIAMATA «ITERAZIONE».

SE LA CONDIZIONE NON DIVENTA MAI FALSA, IL CODICE CONTINUA ALL'INFINITO O FINCHÈ NON SI TERMINA IL PROCESSO.

```
1  let index : number = 0;
2  let sum : number = 0;
3
4  while (index < 10) {
5      index++;
6      sum += index;
7  }
8
9  console.log(sum);
10 // 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55
11 |
12
13 let anotherIndex : number = 0;
14 let anotherSum : number = 0;
15
16 while (anotherIndex < 10) {
17     anotherSum += anotherIndex;
18 }
19 // anotherIndex is never changed, so the loop never exits
20
21 console.log(sum);
```

CONTROL FLOW - 2.2



NEL «do...while» INVECE IL «LOOP BODY» È ESEGUITO UNA PRIMA VOLTA, QUINDI LA CONDIZIONE VIENE CONTROLLATA ED EVENTUALMENTE SI RIPETE IL «LOOP BODY» FINCHÈ LA CONDIZIONE NON DIVENTA FALSE.

```
1  let result :string  = '';
2  let index :number  = 0;
3
4  do {
5      index = index + 1;
6      result += index;
7  } while (index < 5);
8
9  console.log(result);
10 // "12345"
```

CONTROL FLOW - 2.3

IL «for» È FORSE IL PIÙ USATO TRA I COSTRUTTI DI LOOP.

OLTRE AL «LOOP BODY» SI INDIVIDUANO TRE ELEMENTI:

- INIZIALIZZAZIONE DELLE VARIABILI (COUNTER, INDEX,...)
- CONDIZIONE DA VALUTARE
- STEP DA ESEGUIRE A FINE LOOP

```
1  /*  
2   for (init; condition; step) {  
3     loop body  
4   }  
5  
6   evaluate init =>  
7     => evaluate condition => if true, execute loop body, then execute step =>  
8       => evaluate condition => if true, execute loop body, then execute step =>  
9         ...  
10        => evaluate condition => if false, exit loop  
11 */  
12  
13 let result :number = 0;  
14 for(let i :number = 1; i < 4; i++) {  
15   result += i**i;  
16 }  
17 console.log(result);  
18 // 1^1 + 2^2 + 3^3 = 1 + 4 + 27 = 32
```



CONTROL FLOW - 2.4

È POSSIBILE USCIRE DA UN LOOP USANDO LA KEYWORD «`break`». CON «`continue`» INVECE È POSSIBILE SALTARE PARTE DEL «LOOP BODY» E RITORNARE ALLA VALUTAZIONE DELLA CONDIZIONE.

LE DUE KEYWORD SI POSSONO USARE IN CICLI «`for`» INNESTATI SE SONO «LABLED».

```
1  let i :number = 0;
2  while (i < 6) {
3      if (i === 3) {
4          break;
5      }
6      i += 1;
7  }

8
9  console.log(i);
10 // 3

11
12 let text :string = '';
13 for (let i :number = 0; i < 10; i++) {
14     if (i % 2 === 0) {
15         continue;
16     }
17     text += i;
18 }

19
20 console.log(text);
21 // "13579"
```

CONTROL FLOW - 2.5

IL CICLO «for» PUÒ ESSERE APPLICATO ANCHE SU OGGETTI E ARRAY:

- SUGLI OGGETTI SI USA
«`for ('variable' in 'object')`»
DOVE «`variable`» È UNA PROPRIETÀ DELL'OGGETTO
- SUGLI ARRAY SI USA
«`for ('variable' of 'iterable')`»
DOVE «`variable`» È L'I-ESIMO ELEMENTO DELL'ARRAY

```
1  const items : unknown[] = Array.from(
2    arrayLike: { length: 100 },
3    mapfn: (_, i : number ) => i + 1
4  );
5  const tests : [{pass: (function(any): boolean): boolean} ] = [
6    { pass: (item) : boolean => item % 2 === 0 },
7    { pass: (item) : boolean => item % 3 === 0 },
8    { pass: (item) : boolean => item % 5 === 0 },
9  ];
10 let itemsPassed : number = 0;
11
12 itemIteration: for (const item of items) {
13   for (const test : {pass: function(any): boolean} of tests) {
14     if (!test.pass(item)) {
15       continue itemIteration;
16     }
17   }
18   itemsPassed++;
19 }
20
21 console.log(itemsPassed)
22 // 3
```

CONTROL FLOW - 3



ESERCIZIO

SCRIVERE UNO SCRIPT CHE

- GENERI UN NUMERO CASUALE DA 1 A 100 USANDO
«`Math.floor(Math.random() * 100) + 1`»
- CHIEDA ALL'UTENTE DI INDOVINARE IL NUMERO GENERATO, TRACCIANDO QUANTI TENTATIVI ESEGUE;
- SE IL NUMERO INSERITO È MAGGIORE O MINORE VISUALIZZA UN MESSAGGIO DI AIUTO;
- SE IL NUMERO È CORRETTO VISUALIZZA UN MESSAGGIO E IL NUMERO TOTALE DI TENTATIVI.

USARE «`parselnt`» PER LA CONVERSIONE IN NUMERO DA «`prompt`».

NON INSERIRE TESTO.

TEMPO: 20 MINUTI

CONTROL FLOW - 4



ESERCIZIO

RISCRIVERE L'ESERCIZIO PRECEDENTE: SE È STATO USATO IL «while»
USARE IL «do.. while» E VICEVERSA.

TEMPO: 15 MINUTI

CONTROL FLOW - 5



ESERCIZIO

SCRIVERE UNO SCRIPT CHE, DATO UN NUMERO IN INPUT, NE CALCOLI IL SUO FATTORIALE, DEFINITO COME IL PRODOTTO DEI NUMERI INTERI POSITIVI MINORI O UGUALI AD ESSO (PER CONVENZIONE $0! = 1$).

SE IL NUMERO INSERITO È NEGATIVO, INVERTIRE IL SEGNO.
NON INSERIRE TESTO.

TEMPO: 20 MINUTI

CONTROL FLOW - 6



ESERCIZIO

PARTENDO DALLO SKETCH A DESTRA,
SCRIVERE UNO SCRIPT CHE, DATO UN
NUMERO n IN INPUT, CALCOLI I PRIMI n
NUMERI DELLA SEQUENZA DI FIBONACCI,
DEFINITA COME LA SEQUENZA IN CUI OGNI
NUMERO È DATO DALLA SOMMA DEI DUE
NUMERI PRECEDENTI.

I VALORI DI PARTENZA SONO 0 E 1;
PER DEFINIZIONE, $\text{FIB}(0) = 1$.

```
// Function to start the Fibonacci sequence generator
function startFibonacciSequence() : string | undefined { Show usages

    // ask user for input and store value

    // Validate the input
    if /* input is invalid: write a condition */ {
        return "Invalid input. Please enter a valid number of terms greater than or equal to 1.";
    }

    // calculate and show the sequence value
}

// Start the Fibonacci sequence generator
startFibonacciSequence();
```

TEMPO: 30 MINUTI

CONTROL FLOW - 7



ESERCIZIO

SCRIVERE UNO SCRIPT CHE, DATO UN NUMERO n IN INPUT,
RESTITUISCA I NUMERI PRIMI FINO A n .
SE n È NEGATIVO, INVERTIRE IL SEGNO.

SUGGERIMENTI:

- UTILIZZARE DUE «for» INNESTATI, UNO PER CICLARE DA 1 A n E UNO PER TROVARE I DIVISORI;
- USARE UNA LABEL SUL CICLO ESTERNO E USARE «continue» PER INTERROMPERE IL CICLO SE IL NUMERO NON È DIVISORE.

TEMPO: 45 MINUTI

FUNZIONI - 1



A LIVELLO DI PROGRAMMAZIONE STRUTTURATA, UNA «FUNZIONE» È UN ESEMPIO DI SUBROUTINE, OVVERO UN «SOTTOPROGRAMMA» CHE PUÒ ESSERE RICHIAMATO DA CODICE ESTERNO ALLA FUNZIONE.

UNA FUNZIONE È COSTITUITA DA UNA SEQUENZA DI STATEMENTS DETTA «FUNCTION BODY»;
PUÒ ACCETTARE PARAMETRI IN INPUT E PUÒ RESTITUIRE UN VALORE IN OUTPUT TRAMITE LA KEYWORD «return».

IN JAVASCRIPT LE FUNZIONI SONO «FIRST-CLASS CITIZENS».

FUNZIONI - 2

PRIMA DI ESSERE RICHIAMATA, UNA FUNZIONE VA DEFINITA.
SI UTILIZZA, IN ORDINE:

- LA KEYWORD «function» CON IL NOME DELLA FUNZIONE;
- UNA LISTA DI PARAMETRI (OPZIONALI) SEPARATI DA VIRGOLE E RACCHIUSE TRA () ;
- GLI STATEMENTS RACCHIUSI TRA {}.

PER RICHIAMARE LA FUNZIONE SI USA IL NOME SEGUITO DA () CON EVENTUALI ARGOMENTI.

```
1  function sum(firstOperand, secondOperand) { Show usages
2      return firstOperand + secondOperand;
3  }
4
5  console.log(sum(1, 2));
6  // 3
7
8  function messageLogger(message) : void { Show usages
9      if (message){
10          console.log(`Message: ${message} , Date.now());
11      }
12  }
13
14 messageLogger('Hello!');
15 // Message: Hello! 1714913184393
```



FUNZIONI - 3



ALCUNE INFORMAZIONI IMPORTANTI:

- I NOMI DELLE FUNZIONI SONO SEMPRE IN CAMEL-CASE; UNA FUNZIONE SENZA NOME È DETTA «ANONIMA»;
- I PARAMETRI IN INPUT SONO PASSATI «*BY VALUE*», QUINDI SE SI RIASSEGNA UN VALORE IL CAMBIAMENTO NON SI PROPAGA ALL'ESTERNO DELLA FUNZIONE.
GLI OGGETTI / ARRAY INVECE SONO PASSATI «*BY SHARING*», QUINDI MUTANDO UNA PROPERTY / UN ELEMENTO, QUESTO CAMBIA ANCHE ALL'ESTERNO DELLA FUNZIONE.

FUNZIONI - 4



- UNA FUNZIONE PUÒ RESTITUIRE UN VALORE IN OUTPUT UTILIZZANDO LA KEYWORD «return»; SE NON C’È IL VALORE RESTITUITO È «undefined»;
- IL «return» SI PUÒ UTILIZZARE PER USCIRE DALLA FUNZIONE;
- PER RESTITUIRE PIÙ VALORI SI POSSONO USARE GLI ARRAY O GLI OGGETTI;
- UNA FUNZIONE PUÒ ESSERE RICHIAMATA ALL’INTERNO DEL SUO «FUNCTION BODY» (FUNZIONE RICORSIVA);
- IL PROTOTIPO DELLE FUNZIONI È L’OGGETTO «Function».

FUNZIONI - 5

```
1  let a : number = 1;
2  console.log(a);
3  // 1
4
5  ⏷ function sumTen(value) : void { Show usages
6      value += 10;
7  }
8  sumTen(a);
9  console.log(a);
10 // 1
11
12 ⏷ function returnSumTen(value) { Show usages
13     return value + 10;
14 }
15 a = returnSumTen(a);
16 console.log(a);
17 // 11
```

```
1  ⏷ const car : {...} = {
2      brand: "Honda",
3      model: "Accord",
4      year: 1998,
5  };
6
7  console.log(car.brand);
8  // Honda
9
10 ⏷ function updateBrand(obj) : void { Show usages
11     // Mutating the object is visible outside the function
12     obj.brand = "Toyota";
13     // Try to reassign the parameter, but this won't affect
14     // the variable's value outside the function
15     obj = null;
16 }
17 updateBrand(car);
18 console.log(car.brand);
19 // Toyota
```

FUNZIONI - 6

UNA FUNZIONE PUÒ DICHIARARE VARIABILI NEL SUO «FUNCTION BODY»; ESSE PERÒ SARANNO VISIBILI SOLO ALL'INTERNO DELLA FUNZIONE STESSA.

SE UNA VARIABILE INTERNA HA LO STESSO NOME DI UNA ESTERNA, PREVALE L'INTERNA («SHADOWING»).

```
1  const name : string = 'Davide';
2
3  function showName() : void { Show usages
4      const name : string = 'Edivad';
5      const surname : string = 'Mininni';
6      console.log(name);
7  }
8
9  showName();
10 // Edivad
11 console.log(name);
12 // Davide
13 console.log(surname);
14 // Uncaught ReferenceError: surname is not defined
```



FUNZIONI - 7

UNA FUNZIONE CHE NECESSITA DI PARAMETRI PUÒ ESSERE ANCHE RICHIAMATA SENZA DI ESSI: IN TAL CASO, ESSI AVRANNO VALORE «*undefined*» (NB: GESTIRE L'ERRORE!).

È POSSIBILE DEFINIRE UN VALORE DI DEFAULT USANDO L'OPERATORE «*=*».

```
1  function logger(message) : void { Show usages
2      console.log(`Message: ${message}`, Date.now());
3  }
4
5  logger();
6  // undefined 1714913184393
7
8  function messageLogger(message : string = 'Unknown error.') : void { Show usages
9      console.log(`Message: ${message}`, Date.now());
10 }
11
12 messageLogger();
13 // Unknown error. 1714913184393
14
15 messageLogger('Hello world!');
16 // Hello world! 1714913184393
```

FUNZIONI - 8



BEST PRACTICES:

- UNA FUNZIONE DOVREBBE SVOLGERE SOLO UN'AZIONE.
UNA FUNZIONE CHE ESEGUE TANTE AZIONI ANDREBBE DIVISA IN PIÙ
FUNZIONI SPECIFICHE PER CIASCUNA AZIONE (NON SEMPRE È POSSIBILE).
- UNA FUNZIONE DOVREBBE AVERE UN NOME CORTO E SIGNIFICATIVO CHE
SI RIFERISCE ALL'AZIONE CHE LA FUNZIONE SVOLGE.
È SUGGERITO USARE PREFISSI:
 - «calc...» SE LA FUNZIONE SVOLGE UN CALCOLO;
 - «get...» / «set...» SE LA FUNZIONE RITORNA O ASSEGNA UN VALORE;
 - «create...» SE LA FUNZIONE GENERA QUALCOSA;
 - ...

FUNZIONI - 9

```
1  function getAge() :number { Show usages
2      return Number(prompt( message: "What's your age?"));
3  }
4
5  function logAllowedAccess(message, age) :void { Show usages
6      console.log(` ${message}, age: ${age}`)
7  }
8
9  function logNotAllowedAccess(message, age) :void { Show usages
10     console.error(` ${message}, age: ${age}`)
11 }
12
13 function allowAccess(age) :boolean { Show usages
14     return age > 18;
15 }
16
17 function navigateToSite() :void { Show usages
18     const age :number = getAge();
19
20     if (allowAccess(age)) {
21         logAllowedAccess( message: 'OK', age);
22     } else {
23         logNotAllowedAccess( message: 'KO', age);
24     }
25 }
26
27 navigateToSite();
```

FUNZIONI - 10



ESERCIZI

RISCRIVERE TUTTI GLI ESERCIZI VISTI FINORA UTILIZZANDO LE FUNZIONI.

TEMPO MASSIMO: 45 - 60 MINUTI

FUNZIONI - 11



TUTTE LE FUNZIONI VISTE FINORA SONO STATE SCRITTE IN MODO DICHIARATIVO («FUNCTION DECLARATION»).

ESISTE UN'ALTRA SINTASSI DI CREAZIONE DELLE FUNZIONI CHIAMATA «FUNCTION EXPRESSION», CHE PERMETTE DI CREARE FUNZIONI IN QUALSIASI PUNTO DEL CODICE. LA SINTASSI È:

- «let» / «const» SEGUITO DAL NOME DELLA VARIABILE
- SEGNO «=»
- «function» SEGUITO NOME DELLA FUNZIONE
- «()» CON EVENTUALI PARAMETRI
- «FUNCTION BODY» TRA PARENTESI GRAFFE

FUNZIONI - 12

IL NOME DELLA FUNZIONE È UTILE SOLO IN CASO DI RICORSIONE; SE NON È PRESENTE SI PARLA DI »FUNZIONE ANONIMA».

LA «FUNCTION EXPRESSION» È CONVENIENTE QUANDO SI DEVONO PASSARE FUNZIONI COME PARAMETRI DI ALTRE FUNZIONI (ES. ARRAY METHODS).

```
1  ⚡function myFunctionDecl() :void { Show usages
2      console.log('Declaration!')
3  }
4
5  ⚡const myFunctionExpr = function () :void { Show usages
6      console.log('Expression!')
7  }
8
9  myFunctionDecl();
10 // Declaration
11 myFunctionExpr();
12 // Expression
13
14 let anotherFunction = myFunctionExpr;
15
16 anotherFunction();
17 // Expression
```



FUNZIONI - 13



UNA FUNZIONE CREATA TRAMITE «DECLARATION» PUÒ ESSERE RICHIAMATA ANCHE PRIMA DI ESSERE DEFINITA; QUESTO PERCHÉ QUANDO L'ENGINE SI PREPARA PER LANCIARE LO SCRIPT, UNA DELLE PRIME COSE CHE FA È RECUPERARE E CREARE LE FUNZIONI GLOBALI DICHIARATE.

AL CONTRARIO, UNA FUNZIONE CREATA TRAMITE «EXPRESSION» PUÒ ESSERE RICHIAMATA SOLO DOPO CHE È STATA ASSEGNATA, OVVERO SOLO DOPO CHE L'ENGINE RAGGIUNGE LA FUNZIONE.

FUNZIONI - 14



UN'ALTRA DIFFERENZA RIGUARDA LO «SCOPE»:

UNA «FUNCTION DECLARATION» È VISIBILE E RICHIAMABILE SOLO NEL BLOCCO DOVE È DICHIARATA, MENTRE UNA «FUNCTION EXPRESSION» PUÒ ESSERE ASSEGNATA A VARIABILI GLOBALI, OVVERO ESTERNAMENTE AL CONTESTO DELLA FUNZIONE.

IN GENERALE, A MENO DI USI SPECIFICI (ES. CALLBACK), LA «FUNCTION DECLARATION» È PREFERIBILE.

FUNZIONI - 15

```
1  let age :string = prompt( message: "What is your age?", _default: 18);
2
3  // conditionally declare a function
4  if (age < 18) {
5      function welcome() :void { Show usages
6          alert("Hello!");
7      }
8  } else {
9      function welcome() :void { no usages
10         alert("Greetings!");
11     }
12 }
13
14 welcome();
// Error: welcome is not defined
```

```
1  let age :string = prompt( message: "What is your age?", _default: 18);
2  let welcome;
3  if (age < 18) {
4      welcome = function() :void {
5          alert("Hello!");
6      };
7  } else {
8      welcome = function() :void {
9          alert("Greetings!");
10     };
11 }
12
13 welcome();
```

FUNZIONI - 16

È POSSIBILE DEFINIRE UNA FUNZIONE ANONIMA IN UN'ALTRA MODALITÀ DETTA «ARROW FUNCTION».

LA SINTASSI PREVEDE LA RIMOZIONE DELLA KEYWORD «function» E L'UTILIZZO DEL SIMBOLO FRECCIA => PRIMA DEL «FUNCTION BODY».

Arrow functions



```
const add = (x, y) => x + y;
```

Normal functions



```
const add = function(x, y) {  
    return x + y;  
};
```

FUNZIONI - 17

CON UN SOLO PARAMETRO, LE TONDE SI POSSONO RIMUOVERE; SE NON CI SONO PARAMETRI INVECE, LE TONDE SONO NECESSARIE. SE LA FUNZIONE CONTIENE PIÙ «STATEMENTS» VANNO RACCHIUSI IN GRAFFE CON UN «return».

SEBBENE ABBIA ALCUNE LIMITAZIONI, QUESTA SINTASSI È MOLTO UTILE IN CASI SPECIFICI (ARRAY, PROMISES, setTimeout...)

```
1  let sumFunc = function(a, b) { Show usages
2      return a + b;
3  }
4
5  let sumArrow = (a, b) => a + b; Show usages
6
7  console.log(sumFunc( a: 1, b: 2));
8 // 3
9  console.log(sumArrow( a: 1, b: 2));
10 // 3
```



FUNZIONI - 18

UNA «CALLBACK» È UNA FUNZIONE CHE VIENE PASSATA COME ARGOMENTO DI UN'ALTRA FUNZIONE E CHE VIENE RICHIAMATA PER ESEGUIRE AZIONI SPECIFICHE.

PRIMA DELLE PROMISES (2012) LE CALLBACKS ERANO L'UNICO MODO PER SCRIVERE CODICE ASINCRONO IN JS.



FUNZIONI - 19

```
1 // sync
2 const log = (message) :void => console.log(message); Show usages
3 const sum = (a, b, callback) :void => { Show usages
4   let total = a + b;
5   callback(total);
6 }
7 sum( 1, 2, log);
8
9 // async
10 let data;
11 function fetchData() :void { Show usages
12   setTimeout( handler: () :{id:number, name:string} => data = {id: 1, name: "John"}, timeout: 1000);
13 }
14
15 fetchData();
16 console.log(data);
17 // undefined
```

```
1 function fetchData(callback) :void { Show usages
2   setTimeout( handler: () => callback(null, {id: 1, name: "John"}), timeout: 1000);
3 }
4 function processData(error, data) :void { Show usages
5   error ? console.error("Error:", error) : console.log("Data:", data);
6 }
7 fetchData(processData);
8
9 function fetchRealData(callback) :void { Show usages
10   fetch( input: 'https://jsonplaceholder.typicode.com/posts/1' ) Promise<Response>
11     .then(response :Response => response.json()) Promise<any>
12     .then(data => callback(null, data)) Promise<any>
13     .catch(error => callback(error, null));
14 }
15 function processRealData(error, data) :void { Show usages
16   error ? console.error("Error:", error) : console.log("Data:", data);
17 }
18 fetchRealData(processRealData);
```

FUNZIONI - 20

UNA FUNZIONE CHE RICHIAMA SE STESSA È DETTA «FUNZIONE RICORSIVA». QUESTO PATTERN È UTILE QUANDO SI PUÒ SUDDIVIDERE UN PROBLEMA COMPLESSO IN PROBLEMI DELLO STESSO TIPO MA PIÙ SEMPLICI.

IN JS, LA RICORSIONE È LIMITATA DALLA GRANDEZZA DELLO STACK DI MEMORIA DISPONIBILE.



FUNZIONI - 21



ESERCIZIO

USANDO UNA FUNZIONE RICORSIVA, RISCRIVERE LO SCRIPT CHE CALCOLA IL FATTORIALE DI UN NUMERO n DATO IN INPUT.

RICORDA:

$$F(0) = 1$$

$$F(n) = n * F(n - 1)$$

ESERCIZIO

USANDO UNA FUNZIONE RICORSIVA, SCRIVERE UNO SCRIPT CHE CALCOLA L' n -ESIMO NUMERO DELLA SEQUENZA DI FIBONACCI, CON n FORNITO IN INPUT.

RICORDA:

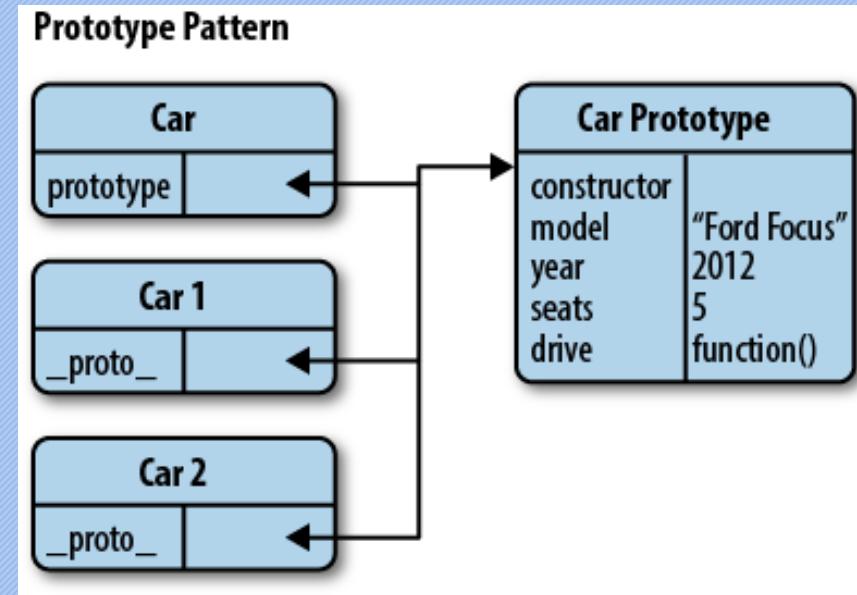
$$\text{FIB}(0) = \text{FIB}(1) = 1;$$

$$\text{FIB}(n) = \text{FIB}(n - 2) + \text{FIB}(n - 1);$$

PROTOTIPO - 1

NELLA PROGRAMMAZIONE IL CONCETTO DI EREDITARIETÀ SI RIFERISCE ALLA CREAZIONE DI ENTITÀ (CHILD/SUB) A PARTIRE DA ALTRE (PARENT/SUPER), IN MODO TALE DA POTER SFRUTTARE IL CODICE GIÀ PRESENTE.

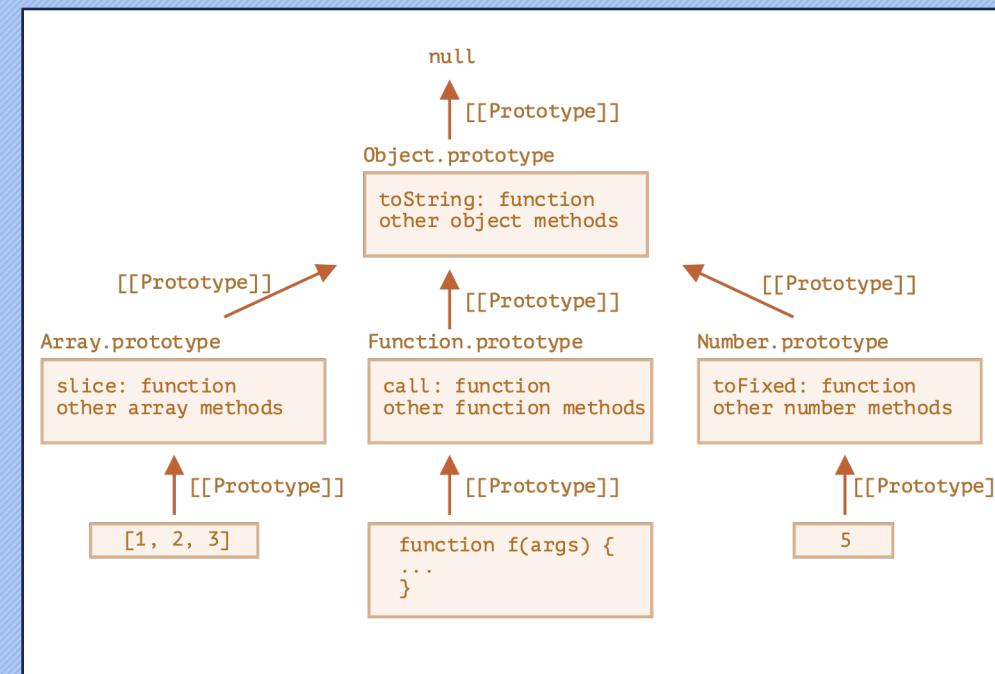
JAVASCRIPT IMPLEMENTA QUESTO CONCETTO TRAMITE GLI OGGETTI: ABBIAMO VISTO CHE OGNI OGGETTO È COLLEGATO AD UN ALTRO TRAMITE «PROTOTIPO».



PROTOTIPO - 2

IL «PROTOTIPO» FUNGE DA ‘MODELLO’ PER TUTTI GLI OGGETTI CREATI A PARTIRE DA ESSO.

IL PROTOTIPO DI BASE È «null» E DA ESSO SI COSTRUISCE POI LA «PROTOTYPE CHAIN». SE UNA PROPERTY NON C’È SULL’OGGETTO, SI RISALE A CERCARLA LUNGO LA «CHAIN».



PROTOTIPO - 3

PER SETTARE IL PROTOTIPO SU UN OGGETTO SI POSSONO DUE METODI:

- POSSIAMO SETTARE DIRETTAMENTE LA PROPERTY «`__proto__`» SULL'OGGETTO;
- POSSIAMO USARE UNA FUNZIONE COSTRUTTORE CON «`new`». QUESTA È LA PRASSI LAVORANDO CON GLI OGGETTI.

```
1  let animal :{eats: boolean, walk: function(): void} = {
2    eats: true,
3    walk() :void {
4      alert("Animal walk");
5    }
6  };
7  let rabbit :{jumps: boolean} = {
8    jumps: true
9  };
10 rabbit.__proto__ = animal;
11
12 let longEar :{__proto__: {...}, earLength: number} = {
13   earLength: 10,
14   __proto__: rabbit
15 };
16
17 // we can find both properties in rabbit now:
18 alert( rabbit.eats ); // true
19 alert( rabbit.jumps ); // true
20 rabbit.walk(); // Animal walk
21
22 // walk is taken from the prototype chain
23 longEar.walk(); // Animal walk
24 alert(longEar.jumps); // true (from rabbit)
```

OGGETTI - 1

IN JAVASCRIPT, GLI OGGETTI SONO UNO DEI MODI POSSIBILI PER MEMORIZZARE DATI COMPLESSI.

UN OGGETTO È UNA COLLEZIONE DI PROPRIETÀ, E UNA PROPRIETÀ È UN ASSOCIAZIONE TRA UN NOME («CHIAVE») E UN VALORE.

IL VALORE DI UNA PROPRIETÀ PUÒ ESSERE DI QUALUNQUE TIPO; SE È UNA FUNZIONE ALLORA LA PROPRIETÀ È DETTA «METODO».

Java: I am oop language
C++: I am oop language
JavaScript:



OGGETTI - 2

IL PROTOTIPO DI TUTTI GLI OGGETTI È «Object».
CI SONO TRE MODI PER CREARE UN OGGETTO:

- TRAMITE «OBJECT INITIALIZER»;
- USANDO UNA FUNZIONE COSTRUTTORE E LA KEYWORD «new»;
- USANDO «Object.create(...)» o «new Object()»;

```
1 // OBJECT INITIALIZER
2 const firstCar :{...} = {
3   brand: "Honda",
4   model: "Accord",
5   year: 1998,
6 };
7
8 // CONSTRUCTOR FUNCTION
9 function CarConstructor(brand, model, year) :void { Show usages
10   this.brand = brand;
11   this.model = model;
12   this.year = year;
13 }
14 const secondCar :CarConstructor = new CarConstructor( brand: "Honda", model: "Accord", year: 1998);
15
16 // USING Object.create()
17 const CarProto :{...} = {
18   brand: "Honda",
19   model: "Accord",
20   year: 1998,
21 };
22 const thirdCar :... = Object.create(CarProto);
```



OGGETTI - 2.1

«`this`» È UNA KEYWORD DI JAVASCRIPT.

POSSIAMO PENSARLO COME UN ‘PARAMETRO NASTCOSTO’ CHE SI RIFERISCE AL CONTESTO DOVE IL CODICE VIENE ESEGUITO (E NON DOVE È DEFINITO).

NEI METODI ESSO FA SEMPRE RIFERIMENTO ALL’OGGETTO SU CUI IL METODO È CHIAMATO.

```
1  function sayHello() :void { Show usages
2      console.log(`Hello, my name is ${this.name}`)
3  }
4
5  const user :{introduce: sayHello, name: string} = {
6      name: 'John',
7      introduce: sayHello
8  }
9
10 const admin :{introduce: sayHello, name: string} = {
11     name: 'Mike',
12     introduce: sayHello
13 }
14
15 user.introduce();
16 // Hello, my name is John
17 admin.introduce();
18 // Hello, my name is Mike
```

OGGETTI - 3



«OBJECT INITIALIZER»:

- LE PROPERTY SONO STRINGHE O SYMBOLS; SE LA PROPERTY È COMPOSTA DA DUE O PIÙ PAROLE, DEVONO ESSERE MESSE TRA APICI;
- LE PROPRIETÀ CALCOLATE VANNO MESSE TRA PARENTESI QUADRE;
- È POSSIBILE USARE IL «REST OPERATOR» PER AGGIUNGERE PROPERTY DA UN ALTRO OGGETTO («SPREAD SYNTAX»);
- È POSSIBILE «IMPORTARE» PROPERTY GIÀ DEFINITE.

OGGETTI - 4

```
1  | const obj1 :{prop1:string, prop2:string} = { prop1: '1', prop2: '2' };
2  |
3  | const shorthand :string = '3';
4  | const shorthand2 :{a:string, b:string} = { a: 'aaa', b: 'bbb' };
5  | const obj2 :{shorthand:string, shorthand2:{...}} = { shorthand, shorthand2 };
6  |
7  | const obj3 :{...} = {
8  |   'another prop': 1000,
9  |   [shorthand2.a]: "key is aaa",
10 |   ...obj1,
11 |   ...obj2
12 | };
13 |
14 | console.log(obj3);
15 | // aaa: "it's 1"
16 | // another prop: 1000
17 | // prop1: "1"
18 | // prop2: "2"
19 | // shorthand: "3"
20 | // shorthand2:
21 | //   a: "aaa"
22 | //   b: "bbb"
```

OGGETTI - 5



FUNZIONE COSTRUTTORE:

- PER CONVENZIONE, IL NOME INIZIA CON LA MAIUSCOLA;
- SI USA IL «this» PER ASSEGNAME LE PROPERTY DELL'OGGETTO;
- L'ORDINE DEI PARAMETRI E DEGLI ARGOMENTI DEVE ESSERE LO STESSO;
- È POSSIBILE ASSEGNAME NUOVE PROPERTY DOPO CHE L'OGGETTO È STATO CREATO; CIÒ PERÒ VALE PER LA SINGOLA ISTANZA E NON MODIFICA IL COSTRUTTORE.
PER AGGIUNGERE PROPERTY GLOBALI SI PUÒ MODIFICARE IL COSTRUTTORE TRAMITE PROTOTYPE.

OGGETTI - 6

```
1  function Car(brand, model, year) : void { Show usages
2      this.brand = brand;
3      this.model = model;
4      this.year = year;
5  }
6  const car :Car = new Car( brand: "Honda", model: "Accord", year: 1998);
7
8  const person :{firstName: string, lastName: string} = {
9      firstName: 'Davide',
10     lastName: 'Mininni',
11 }
12 car.owner = person;
13
14 console.log(car.owner.firstName);
15 // Davide
```

OGGETTI - 7



«`Object.create(...)`»:

- MENO USATO DEI PRECEDENTI;
- UTILE PERCHÉ PERMETTE DI DEFINIRE IL PROTOTIPO DELL'OGGETTO SENZA DOVER SCRIVERE UNA FUNZIONE COSTRUTTORE.

«`new Object()`»:

- POCO USATO, LE PROPERTY VANNO AGGIUNTE UNA AD UNA;
- IL PROTOTIPO È «`Object`».

OGGETTI - 8



ESERCIZIO

CREARE UNA FUNZIONE COSTRUTTORE PER UN CONTATTO DELLA RUBRICA («Contact»).

CAMPI OBBLIGATORI: NOME, EMAIL, CELLULARE.

CAMPI FACOLTATIVI: A PIACERE.

AGGIUNGERE TRAMITE PROTOTIPO UNA FUNZIONE CHE RITORNI UNA STRINGA CON TUTTI I DETTAGLI

CREARE UN CONTATTO E VISUALIZZARE I DETTAGLI IN UN ALERT.

TEMPO: 15 MINUTI

OGGETTI - 9

LE PROPERTY DI UN OGGETTO,
SE NON SONO SYMBOLS, SONO
SEMPRE CONVERTITE IN
STRINGHE.

PER ACCEDERE O SETTARE LE
PROPERTY SI POSSONO USARE
DUE MODI:

- IL PUNTO .
- LE QUADRE: NECESSARIE IN CASO
DI PAROLE COMPOSTE

```
1 const person :{...} = {  
2   name: {  
3     firstName: 'Davide',  
4     lastName: 'Mininni'  
5   },  
6   age: 34,  
7   "hair's color": 'black',  
8 }  
9  
10 console.log(person.age);  
11 // 34  
12 console.log(person.name.firstName);  
13 // Davide  
14 console.log(person["hair's color"]);  
15 // black  
16  
17 person.sex = 'm';  
18 person["eye's color"] = 'brown';
```

OLTRE AL VALORE, LE PROPERTY HANNO TRE DIVERSI FLAG:

- «`writable`»
SE «`true`», IL VALORE PUÒ ESSERE CAMBIATO, SE «`false`» LA PROPERTY È DI SOLA LETTURA;
- «`enumerable`»
SE «`true`», IL VALORE VIENE VISUALIZZATO NEI CICLI (es. «`for...in`»);
- «`configurable`»
SE «`true`», LA PROPERTY PUÒ ESSERE CANCELLATA E QUESTI STESSI TRE FLAG POSSONO ESSERE MODIFICATI, ALTRIMENTI NO.

«`Object`» ESPONE METODI PER LIMITARE L'ACCESSO ALL'OGGETTO.

PER OTTENERE LE PROPERTY DI UN OGGETTO SI PUÒ USARE:

- IL CICLO «`for.. in`»: L'i-ESIMO ELEMENTO È UNA PROP ENUMERABILE DELL'OGGETTO O DEI PROTOTIPI;
- «`Object.keys()`»: LISTA DELLE PROP ENUMERABILI DELL'OGGETTO MA NON DEI PROTOTIPI;
- «`Object.getOwnPropertyNames()`»: LISTA DELLE PROP DELL'OGGETTO, MA NON DEI PROTOTIPI.

```
1  function Car(brand, model, year) : void { Show usages
2      this.brand = brand;
3      this.model = model;
4      this.year = year;
5  }
6  const car :Car = new Car( brand: "Honda", model: "Accord", year: 1998);
7  car.fuel = 'Diesel';
8  Object.defineProperty(car, p: 'owner', attributes: { value: 'Davide', enumerable: false });
9
10 for (const carKey in car) {
11     console.log(carKey);
12 }
13 // brand
14 // model
15 // year|                                     ← cursor position
16 // fuel
17
18 console.log(Object.keys(car));
19 // ['brand', 'model', 'year', 'fuel']
20
21 console.log(Object.getOwnPropertyNames(car));
22 // ['brand', 'model', 'year', 'fuel', 'owner']
```

OGGETTI - 12



LE PROPERTY NON SONO ORDINATE.

SE LE PROPERTY POSSONO ESSERE CONVERTITE DA/A NUMERO SENZA MODIFICA, ALLORA SARANNO VISUALIZZATE NEI CICLI IN ORDINE CRESCENTE;

SE LA CONVERSIONE NON È POSSIBILE, ALLORA LE PROPERTY SONO VISUALIZZATE IN ORDINE DI CREAZIONE (VEDI LOG DELL'IMMAGINE PRECEDENTE).

PER CANCELLARE UNA PROPERTY SI USA LA KEYWORD «`delete`».

LE PROPERTY VISTE FINORA SONO DETTA «DATA PROPERTY».

ESISTE UN ALTRO TIPO, DETTO «ACCESSOR PROPERTY», CHE SI CONTRADDISTINGUE PER LE KEYWORDS «get» E «set». INTERNAMENTE POSSONO ESSERE VISTE COME FUNZIONI CHE FANNO GET/SET DI VALORI, MENTRE ESTERNAMENTE NON SI DISTINGUONO DALLE PRECEDENTI.

POSSENTI ESSERE USATI AD ESEMPIO QUANDO UNA PROPERTY DEVE RESTITUIRE UN VALORE DINAMICAMENTE CALCOLATO.

OGGETTI - 14

```
1  let user :{...} = {  
2      name: "John",  
3      surname: "Ford",  
4  
5      get fullName():string {  
6          return `${this.name} ${this.surname}`;  
7      },  
8  
9      set fullName(value) {  
10         [this.name, this.surname] = value.split(" ");  
11     }  
12 };  
13  
14 console.log(user.fullName);  
15 // John Ford  
16  
17 user.fullName = "Jane Smith";  
18 console.log(user.name);  
19 // Jane  
20 console.log(user.surname);  
21 // Smith
```

OGGETTI - 15



ESERCIZIO

PARTENDO DALL'ESERCIZIO PRECEDENTE, CREARE UNA RUBRICA («CONTACT MANAGER») CHE ABBIA:

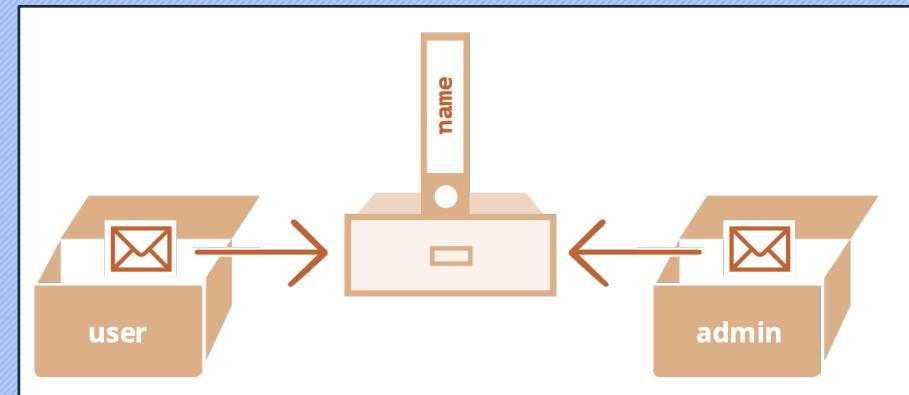
- UNA VARIABILE «contact» DI TIPO OGGETTO PER MEMORIZZARE I CONTATTI (KEY: NOME);
- UN METODO PER AGGIUNGERE E UNO PER ELIMINARE UN CONTATTO
- UN METODO PER TROVARE UN CONTATTO;
- UN GET CHE RITORNA UNA I DETTAGLI DI TUTTI I CONTATTI;
- USARE TUTTI I METODI CREATI.

TEMPO: 45 MINUTI

OGGETTI - 16

A DIFFERENZA DELLE ALTRE PRIMITIVE, GLI OGGETTI SONO COPIATI «*BY REFERENCE*»:
CIÒ SIGNIFICA CHE, IN FASE DI CREAZIONE, L'OGGETTO VIENE MEMORIZZATO E LA VARIABILE ‘PUNTA’ AD ESSO.
COPIARE UN OGGETTO QUINDI NON COPIA IL VALORE, MA COPIA ‘IL PUNTAMENTO’ ALLA LOCAZIONE DI MEMORIA.

```
1 let user :{name: string} = { name: "John" };
2 let admin :{name: string} = user; // copy the reference
```



PER COPIARE UN OGGETTO, SI PUÒ UTILIZZARE

- LO «SPREAD OPERATOR»;
- IL METODO
«`Object.assign(dest, ...source)`», OVE
«`dest`» È L'OGGETTO DI ARRIVO
«`source`» SONO GLI OGGETTI DI
PARTENZA.

```
1  let user :{name: string} = { name: "John" };
2  let admin :{name: string} = user;
3  console.log(user === admin);
4  // true
5
6  admin = {...user};
7  console.log(user === admin);
8  // false
9
10 admin.isSuper = false;
11 const superAdmin :... & ... = Object.assign(admin, {isSuper: true})
12 console.log(superAdmin.name);
13 console.log(superAdmin.isSuper);
14 // John
15 // true
```

OGGETTI - 18



I DUE METODI NON FUNZIONANO IN CASO DI OGGETTI INNESTATI:
QUANDO CLONATO, L'OGGETTO INTERNO MANTIENE IL RIFERIMENTO
ALL'OGGETTO DI PARTENZA.

PER OVVIARE A QUESTO PROBLEMA SI PUÒ USARE
«Object. structuredClone(...)», TUTTAVIA ANCHE QUESTO METODO
FALLISCE SE UN OGGETTO HA DEI PROPERTY CHE SONO FUNZIONI.

PER CASI COMPLESSI, SI USANO LIBRERIE ESTERNE
(ES. lodash/cloneDeep).

ARRAY - 1

ABBIAMO VISTO CHE GLI OGGETTI PERMETTONO DI MANEGGIARE DATI COMPLESSI; TUTTAVIA LE PROPERTY DI UN OGGETTO NON SONO ORDINATE/ORDINABILI.

QUANDO SI HA NECESSITÀ DI AVERE LISTE ORDINATE DI ELEMENTI DELLO STESSO TIPO SI PUÒ UTILIZZARE UN TIPO SPECIALE DI OGGETTO DETTO ARRAY («`typeof []`: Object»).



ARRAY - 2

UN ARRAY È UNA LISTA DI DATI COERENTI DI QUALSIASI NATURA ORDINATI IN BASE AL VALORE DI UN INDICE. L'INDICE È UN NUMERO INTERO POSITIVO; IL PRIMO ELEMENTO HA SEMPRE INDICE «0».

PER CREARE UN ARRAY SI POSSONO USARE LE PARENTESI QUADRE O IL COSTRUTTORE «Array»; SI ACCEDE AGLI ELEMENTI NON COL PUNTO MA CON LE PARENTESI QUADRE.

```
1 const arrayOne :any[] = [];
2 console.log(arrayOne.length);
// 0
3
4
5 const arrayTwo :string[] = ['a', 'b', 'c'];
6 console.log(arrayTwo.length);
// 3
7
8
9 const anotherOne :any[] = new Array( arrayLength: 10);
10 console.log(anotherOne.length, anotherOne[0]);
// 10 undefined
11
12
13 const more :number[] = Array( items: 1, 2);
14 console.log(more.length);
// 2
15
16
17 const complexTypeArray :[{var2: string, var1: number},... ] = [
18   {var1: 1, var2: 'two'},
19   {var1: 10, var2: 'twenty'}
20 ];
21 console.log(complexTypeArray.length);
// 2
22 console.log(complexTypeArray[0]);
// var1: 1, var2: 'two'
23 console.log(complexTypeArray[1].var1);
// 10
24 console.log(complexTypeArray[1].var2);
// twenty
```

ARRAY - 3

LA PROPERTY «length» RESTITUISCE IL NUMERO DI ELEMENTI NELL'ARRAY. POICHÉ L'INDICE PARTE DA ZERO, LA «length» CORRISPONDE ALL'INDICE DELL'ULTIMO ELEMENTO PIÙ UNO.

È POSSIBILE INSERIRE UN ELEMENTO IN UNA POSIZIONE QUALSIASI DELL'ARRAY ASSEGNAVANDO IL VALORE ALL'ELEMENTO NELLA POSIZIONE DESIDERATA; LA «length» SI MODIFICA DI CONSEGUENZA.

```
1  const arr :number[] = [1, 2, 100];
2  console.log(arr.length);
3  // 3
4
5  arr[2] = 999;
6  console.log(arr);
7  // [1, 2, 999]
8  console.log(arr.length);
9  // 3
10
11 arr[3] = 0;
12 console.log(arr.length);
13 // 4
14
15 // "arr" became a "sparse array"
16 arr[5] = 123;
17 console.log(arr.length);
18 // 6
19 console.log(arr);
20 // [1, 2, 999, 0, empty, 123]
```



ARRAY - 4

PER CICLARE SUGLI ELEMENTI DI UN ARRAY, OLTRE AL NORMALE CICLO «for» SI POSSONO USARE:

- «for...of»
- «.forEach()» SULL'ARRAY STESSO

ENTRAMBI ITERANO DIRETTAMENTE SULLA SEQUENZA DEI VALORI CHE COSTITUISCONO L'ARRAY.

```
1  const iterable :number[] = [10, 20, 30];
2  for (const value :number of iterable) {
3      console.log(value);
4  }
5  // 10
6  // 20
7  // 30
8
9  const anotherIterable :string[] = ['aba', 'bcb', 'cdc'];
10 anotherIterable.forEach((element :string, index :number, array :string[] ) :void => {
11     console.log(`The array '${array}' has at index ${index} the element: ${element}`);
12 })
13 // The array 'aba,bcb,cdc' has at index 0 the element: aba
14 // The array 'aba,bcb,cdc' has at index 1 the element: bcb
15 // The array 'aba,bcb,cdc' has at index 2 the element: cdc
```



ARRAY - 5



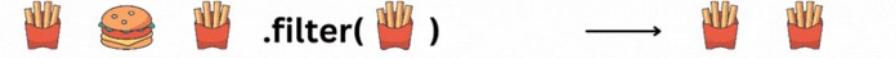
PER TRASFORMARE UN ARRAY (ORDINARE, RIMAPPARE, FILTRARE, AGGIUNGERE E RIMUOVERE ELEMENTI, TROVARE UN ELEMENTO/L'INDICE DI UN ELEMENTO SPECIFICO ECC.) SI POSSONO USARE I METODI DELL'OBJECT WRAPPER «ARRAY».

UNA COSA DA CONSIDERARE È IL TIPO DI RITORNO DELLA FUNZIONE UTILIZZATA: ALCUNE DI ESSE OPERANO CAMBIANDO DIRETTAMENTE L'ARRAY DI PARTENZA, ALTRE CREANO UNA COPIA CON IL RISULTATO DELLA FUNZIONE. FARE SEMPRE RIFERIMENTO A MDN.

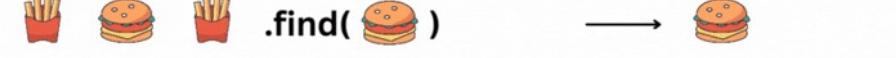
ARRAY - 6



`.map( → )` →   



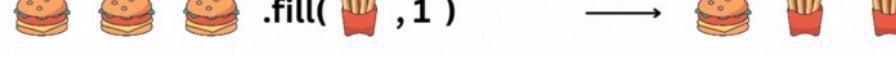
`.filter()` →  



`.find()` → 



`.findIndex()` → `1`



`.fill( , 1)` →   



`.some()` → `true`



`.every()` → `false`

ARRAY - 7



ESERCIZIO

GESTIRE UNA TODO LIST CON TODO DI TIPO STRINGA:

- CREARE UNA FUNZIONE CHE AGGIUNGA UN TODO A UN ARRAY;
- CREARE UNA FUNZIONE CHE ELIMINI UN TODO DALL'ARRAY;
- CREARE UNA FUNZIONE CHE VISUALIZZI TUTTI I TODO;
- TESTARE TUTTE LE FUNZIONI.

USARE LE API: «.indexOf», «.push», «.slice» E «.forEach».

TEMPO: 20 MINUTI

ARRAY - 8



ESERCIZIO

SCRIVERE UNO SCRIPT CHE

- CHIEDA ALL'UTENTE DI INSERIRE IL NUMERO DI ESAMI SOSTENUTI;
- PER CIASCUN ESAME CHIEDA DI INSERIRE IL VOTO OTTENUTO (NUMERICO);
- CALCOLI E VISUALIZZI LA MEDIA ARITMETICA.

USARE

- «.push» PER SALVARE IN UN ARRAY I RISULTATI DEGLI ESAMI;
- «.reduce» PER CALCOLARE LA MEDIA;
- (OPZIONALE: COME SI PUÒ USARE «for..in» / »forEach» PER CICLARE SUL NUMERO DI ESAMI?)

TEMPO: 30 MINUTI

ESERCIZIO

MODIFICARE L'ESERCIZIO DELLA TODO-LIST:

- IL TODO NON È PIÙ UNA STRINGA MA UN OGGETTO CON CAMPI TIPO, NOME (STRINGHE) E COMPLESSITÀ (NUMERO);
- MODIFICARE LA DELETE TENENDO CONTO DEL CAMBIO STRINGA - OGGETTO («.findIndex» / «.splice»);
- AGGIUNGERE FUNZIONI CHE PERMETTANO DI VISUALIZZARE I TODO FILTRANDO PER TIPO E COMPLESSITÀ («.filter»);

TEMPO: 20 MINUTI

ARRAY - 10



ESERCIZIO

CONTINUANDO L'ESERCIZIO PRECEDENTE:

- AGGIUNGERE UNA FUNZIONE CHE VISUALIZZI I TODO ORDINATI PER COMPLESSITÀ («.sort»);
- AGGIUNGERE UNA FUNZIONE CHE MOSTRI UN WARNING SE I TODO SONO IN NUMERO MAGGIORE DI 10 O SE ALMENO UNO HA COMPLESSITÀ 10 («.some»);
- AGGIUNGERE UN CAMPO DURATA AL COSTRUTTORE «Todo» (MINUTI); RIMAPPARE LA DURATA IN ORE E CALCOLARE LA DURATA TOTALE («.map» / «.reduce»);

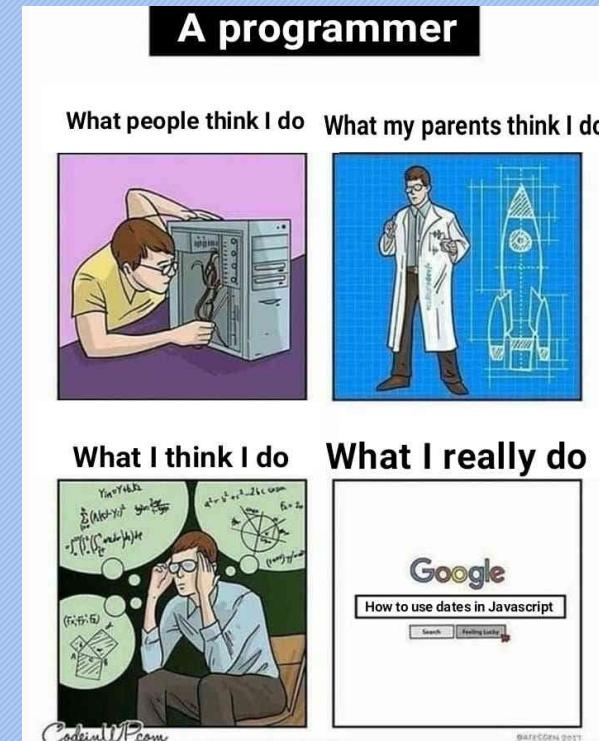
TEMPO: 20 MINUTI

DATE - 1

PER LA GESTIONE DELLE DATE,
JAVASCRIPT METTE A DISPOSIZIONE
L'OGGETTO «Date».

UNA DATA IN JAVASCRIPT È
SPECIFICATA COME IL TEMPO IN
MILLISECONDI CHE È PASSATO DALLA
MEZZANOTTE DEL 01.01.1970 UTC.

L'API NON È PERFETTA; SPESSO SI
FA USO DI LIBRERIE ESTERNE
(ECMASCRIPT È AL LAVORO SU UN
NUOVO STANDARD - «TEMPORAL»)



DATE - 2



PER CREARE UNA DATA SI USA IL COSTRUTTORE «Date()»,
CHE PUÒ ESSERE CHIAMATO:

- SENZA PARAMETRI: RITORNA LA DATA ATTUALE AL MILLISECONDO;
- CON UN NUMERO n: RITORNA LA DATA CORRISPONDENTE ALLA DATA ZERO PIÙ n MILLISECONDI;
- CON UNA STRINGA: PARSA LA STRINGA INTERPRETANDOLA COME DATA;
- CON UNA DATA: GENERA UNA NUOVA ISTANZA CON LA STESSA DATA;
- CON 2–7 PARAMETRI: GENERA UNA DATA IN BASE AI PARAMETRI PASSATI, IN ORDINE: ANNO, MESE, GIORNO, ORA, MINUTO, SECONDO, MILLISECONDO.

DATE - 3

```
1 const date :Date = new Date();
2 console.log(date);
3 // Tue May 07 2024 12:58:40 GMT+0200 (Ora legale dell'Europa centrale)
4 const dateZero :Date = new Date( value: 0 );
5 console.log(dateZero);
6 // Thu Jan 01 1970 01:00:00 GMT+0100 (Ora standard dell'Europa centrale)
7 const dateOneDay :Date = new Date( value: 1000 * 60 * 60 * 24 );
8 console.log(dateOneDay);
9 // Fri Jan 02 1970 01:00:00 GMT+0100 (Ora standard dell'Europa centrale)
10 const dateString :Date = new Date( value: '11-1-20' );
11 console.log(dateString);
12 // Tue Nov 01 2020 00:00:00 GMT+0100 (Ora standard dell'Europa centrale)
13 const dateObject :Date = new Date(new Date( value: "2024-12-17T03:24:00" ));
14 console.log(dateObject);
15 // Tue Dec 17 2024 03:24:00 GMT+0100 (Ora standard dell'Europa centrale)
```

```
1 const month :Date = new Date( year: 1999, monthIndex: 0 );
2 console.log(month);
3 // Fri Jan 01 1999 00:00:00 GMT+0100 (Ora standard dell'Europa centrale)
4
5 const day :Date = new Date( year: 1999, monthIndex: 0, date: 15 );
6 console.log(day);
7 // Fri Jan 15 1999 00:00:00 GMT+0100 (Ora standard dell'Europa centrale)
8
9 const hours :Date = new Date( year: 1999, monthIndex: 0, date: 15, hours: 12 );
10 console.log(hours);
11 // Fri Jan 15 1999 12:00:00 GMT+0100 (Ora standard dell'Europa centrale)
12
13 const minutes :Date = new Date( year: 1999, monthIndex: 0, date: 15, hours: 12, minutes: 30 );
14 console.log(minutes);
15 // Fri Jan 15 1999 12:30:00 GMT+0100 (Ora standard dell'Europa centrale)
16
17 const seconds :Date = new Date( year: 1999, monthIndex: 0, date: 15, hours: 12, minutes: 30, seconds: 15 );
18 console.log(seconds);
19 // Fri Jan 15 1999 12:30:15 GMT+0100 (Ora standard dell'Europa centrale)
20
21 const milliseconds :Date = new Date( year: 1999, monthIndex: 0, date: 15, hours: 12, minutes: 30, seconds: 15, ms: 20000 );
22 console.log(milliseconds);
23 // Fri Jan 15 1999 12:30:35 GMT+0100 (Ora standard dell'Europa centrale)
```

DATE - 4



UNA DATA VIENE RAPPRESENTATA INTERNALEMENTE CON UN TIMESTAMP. IL FUSO ORARIO PERÒ NON È SALVATO; DI CONSEGUENZA QUANDO SI INTERAGISCE CON UNA DATA BISOGNA SAPERE SE FA RIFERIMENTO AL FUSO ORARIO LOCALE O ALLO STANDARD UTC.

L'ISTANTE ZERO AD ESEMPIO È ZERO IN UTC MA PUÒ ESSERE INTERPRETATO COME IL 31/12/1969 ORE 19.00 NEL FUSO DI NY (UTC+5).

PER QUESTO MOTIVO «Date» ESPONE DUE GRUPPI DI METODI: UNO CHE CONSIDERA IL LOCAL TIME E UNO CHE CONSIDERA L'UTC.

DATE - 5

COMPONENTE	LOCAL TIME		UTC	
	GET	SET	GET	SET
ANNO	getFullYear()	setFullYear()	getUTCFullYear()	setUTCFullYear()
MESE	getMonth()	setMonth()	getUTCMonth()	setUTCMonth()
GIORNO (MESE)	getDate()	 setDate()	getUTCDate()	setUTCDate()
ORE	getHours()	setHours()	getUTCHours()	setUTCHours()
MINUTI	getMinutes()	setMinutes()	getUTCMinutes()	setUTCMinutes()
SECONDI	getSeconds()	setSeconds()	getUTCSeconds()	setUTCSeconds()
MILLISECONDI	getMilliseconds()	setMilliseconds()	getUTCMilliseconds()	setUTCMilliseconds()
GIORNO (SETTIMANA)	getDay()	N/A	getUTCDay()	N/A

ALCUNE CONSIDERAZIONI PER LA CREAZIONE:

- È BUONA NORMA FORNIRE GLI ANNI CON 4 CIFRE;
SE SE NE USANO 2, JAVASCRIPT ASSUME SIANO DEL 1900
(«new Date(24, 4)» => MAGGIO 1924);
- I MESI PARTONO DA 0 (GEN: 0, FEB: 1, MAR: 2, ...);
- SE IL GIORNO NON È FORNITO, SI PARTE DAL 1° DEL MESE;
- SE ORE, MINUTI, SECONDI O MILLISECONDI NON SONO FORNITI,
PARTONO DA 0.

DATE - 7

- SE IN CREAZIONE VIENE FORNITA UNA DATA FUORI DAL RANGE, L'API SI «CORREGGE» CREANDO UNA DATA CORRETTA.
- IL CONFRONTO TRA DATE VA SEMPRE FATTO CONSIDERANDO I MILISECONDI.
PER OTTENERE LA DATA CORRENTE IN MS, INVECE DI USARE «`new Date().getTime()`» SI PUÒ USARE «`Date.now()`».

```
1  /*  
2   * monthIndex => December has index 11, so 12 means January of the next year  
3   * date => January has 31 days, so 35 means 31 + 4 = 4 February  
4   * expected: 04.02.2016  
5  */  
6  const date :Date = new Date( year: 2015, monthindex: 12, date: 35);  
7  console.log(date);  
8  // Thu Feb 04 2016 00:00:00 GMT+0100 (Ora standard dell'Europa centrale)  
9  
10 const now :number = new Date().getTime();  
11 const rightNow :number = Date.now();  
12 console.log(now - rightNow);  
13 // 0
```

DATE - 8



ECMASCRIPT PRESCRIBE UN SOLO FORMATO UNIVERSALMENTE SUPPORTATO PER LA CONVERSIONE DA/A STRINGA:

«YYYY-MM-DDTHH:mm:ss.sssZ»

- 4 CIFRE PER GLI ANNI;
- 2 CIFRE PER I MESI (RANGE 01 - 12);
- 2 CIFRE PER I GIORNI (RANGE 01 - 31);
- LA LETTERA «T» È OBBLIGATORIA SE SI STA FORNENDO ANCHE IL TEMPO;
- 2 CIFRE PER LE ORE (RANGE 0 - 23, MA «24:00:00» È ACCETTATO);

ECMASCRIPT PRESCRIBE UN SOLO FORMATO UNIVERSALMENTE SUPPORTATO PER LA CONVERSIONE DA/A STRINGA:

«YYYY-MM-DDTHH:mm:ss.sssZ»

- 2 CIFRE PER I MINUTI (RANGE 00 - 59);
- 2 CIFRE PER I SECONDI (RANGE 00 - 59);
- 3 CIFRE PER I MILLISECONDI (RANGE 000 - 999)
- LA LETTERA «Z» INDICA IL FUSO ORARIO:
 - «Z» INDICA IL TEMPO UTC;
 - PER ALTRI FUSI SI USA +/- SEGUITO DALL'OFFSET IN FORMATO «HH:mm»

ECMASCRIPT PRESCRIBE UN SOLO FORMATO UNIVERSALMENTE SUPPORTATO PER LA CONVERSIONE DA/A STRINGA:

«YYYY-MM-DDTHH:mm:ss.sssZ»

MOLTI ELEMENTI POSSONO ESSERE OMESSI, AD ESEMPIO SONO VALIDI:

- «YYYY» / «YYYY-MM» / «YYYY-MM-DD»;
- UNO DEI TRE PRECEDENTI SEGUITO DALLA LETTERA «T» E SEGUITO DA UNO TRA: «HH:mm» / «HH:mm:ss» / «HH:mm:ss.sss»;
- UNA QUALUNQUE COMBINAZIONE PRECEDENTE SEGUITA DAL FUSO ORARIO.

DATE - 9.3



ECMASCRIPT PRESCRIBE UN SOLO FORMATO UNIVERSALMENTE SUPPORTATO PER LA CONVERSIONE DA/A STRINGA:

«YYYY-MM-DDTHH:mm:ss.sssZ»

QUANDO IL FUSO NON È PRESENTE LE DATE SENZA TEMPO SONO CONSIDERATE UTC, MENTRE QUELLE CON IL TEMPO SONO CONSIDERATE NEL FUSO ORARIO LOCALE.

APPROFONDIMENTO: <https://maggiepint.com/2017/04/11/fixing-javascript-date-web-compatibility-and-reality/>

DATE - 10



ESERCIZIO

CREARE UNA FUNZIONI CHE, FORNITA IN INPUT LA DATA DI NASCITA DI UNA PERSONA, NE CALCOLI L'ETÀ IN ANNI, MESI E GIORNI RISPETTO ALLA DATA ATTUALE.

IMPORTANTE: QUANDO SI CONFRONTANO I TRE VALORI PER IL CALCOLO, AGGIUSTARE IL CONTEGGIO IN BASE ALLE DIFFERENZE NEGATIVE, ES: 31.05.1990 => 22.05.2024

22 - 31 = -9 → TOGLIERE 9 GIORNI AL MESE PRECEDENTE

TEMPO: 30 MINUTI

ESERCIZIO

CREARE UN PIANIFICATORE FERIE:

- DEFINIRE UNA FUNZIONE COSTRUTTORE «Holiday» CHE ABBIA TITOLO, DESCRIZIONE E DATA;
- CREARE UN OGGETTO «HolidayPlanner» CON ALL'INTERNO UNA PROPERTY «holiday» DI TIPO ARRAY;
- AGGIUNGERE FUNZIONI PER AGGIUNGERE E CANCELLARE FERIE;
- AGGIUNGERE UNA FUNZIONE CHE FORNITA UNA DATA RESTITUISCA TUTTE LE FERIE NEI SUCCESSIVI 7 GIORNI;
- AGGIUNGERE UNA FUNZIONE CHE FORNITA UNA DATA RESTITUISCE TUTTE LE FERIE PRECEDENTI.

TEMPO: 20 MINUTI

MAP - 1

UNA «Map» IN JAVASCRIPT È UN OGGETTO CHE PERMETTE DI MEMORIZZARE COPPIE CHIAVE - VALORE.

LE CHIAVI DI UNA MAPPA SONO UNIVOCHE; CICLARE SU UNA MAPPA (CON UN «for..of») RITORNA UN ARRAY DI DUE ELEMENTI [CHIAVE, VALORE] AD OGNI ITERAZIONE.

SE ESISTONO GIÀ GLI OGGETTI, A COSA SERVONO LE MAPPE?

JAVASCRIPT MEMOIZATION



{ }

Map()

MAP - 2



MAPPA	OGGETTO
UNA MAPPA NON HA CHIAVI, TRANNE QUELLE AGGIUNTE DALL'UTENTE.	UN OGGETTO HA UN PROTOTIPO, QUINDI HA DELLE PROPERTY DI DEFAULT.
UNA MAPPA È «SAFE TO USE» CON LE COPPIE CHIAVI-VALORE FORNITE DALL'UTENTE.	UN OGGETTO PUÒ SUBIRE UN ATTACCO CHE VADA A FARE OVERRIDE SUL PROTOTIPO E QUINDI SUBIRE ESECUZIONE DI CODICE REMOTO.
LA CHIAVE DI UNA MAPPA PUÒ ESSERE QUALUNQUE COSE (ANCHE UNA FUNZIONE O UN OGGETTO).	LA CHIAVE DI UN OGGETTO È SEMPRE E SOLO UNA STRINGA.

MAP - 2.1



MAPPA	OGGETTO
LE CHIAVI DI UNA MAPPA SONO ORDINATE RISPETTO ALL'ORDINE DI INSERIMENTO.	L'ORDINE DELLE CHIAVI DI UN OGGETTO È UN MECCANISMO NON AFFIDABILE; NON ESISTE UN'API CHE PERMETTE DI ITERARE TUTTE LE PROPERTY DELL'OGGETTO.
IL NUMERO DI ELEMENTI È RICAVABILE TRAMITE LA PROPERTY «size».	IL NUMERO DI ELEMENTI SI PUÒ RICAVARE DA «Object.keys().length» (POCO EFFICIENTE).
UNA MAPPA È ITERABILE.	UN OGGETTO NON È DIRETTAMENTE ITERABILE; SI PUÒ USARE IL «for...in» O ITERARE SULLE KEYS/ENTRIES.

MAP - 2.2



MAPPA	OGGETTO
UNA MAPPA È OTTIMIZZATA QUANDO SI RICHIEDONO FREQUENTI AGGIUNTE/RIMOZIONI.	UN OGGETTO NON È OTTIMIZZATO QUANDO SI RICHIEDONO FREQUENTI AGGIUNTE/RIMOZIONI.
NESSUN SUPPORTO NATIVO AL PARSING.	SUPPORTO NATIVO AL PARSING USANDO «JSON.parse()».
NESSUN SUPPORTO NATIVO ALLA SERIALIZZAZIONE.	SUPPORTO NATIVO ALLA SERIALIZZAZIONE USANDO «JSON.stringify()».

MAI USARE IL GET/SET DEGLI OGGETTI (. / []) SULLE MAPPE!

MAP - 3

PER CREARE UNA MAPPA SI USA IL COSTRUTTORE «Map», CON I METODI:

- «set» PER AGGIUNGERE UN ELEMENTO;
- «has» PER CONTROLLARE SE L'ELEMENTO ESISTE;
- «get» PER OTTENERE L'ELEMENTO CORRISPONDENTE;
- «delete» PER CANCELLARE L'ELEMENTO;
- «clear» PER SVUOTARE LA MAPPA.

```
1 const contacts : Map<any, any> = new Map();
2 // adds an element
3 contacts.set("Jessie", { phone: "213-555-1234", address: "123 N 1st Ave" });
4 // checks for elements
5 contacts.has( key: "Jessie"); // true
6 contacts.has( key: "Hilary"); // false
7 // retrieves elements
8 contacts.get("Hilary"); // undefined
9 contacts.set("Hilary", { phone: "617-555-4321", address: "321 S 2nd St" });
10 contacts.get("Jessie"); // {phone: "213-555-1234", address: "123 N 1st Ave"}
11 // deletes elements
12 contacts.delete( key: "Raymond"); // false
13 contacts.delete( key: "Jessie"); // true
14 //check size
15 console.log(contacts.size); // 1
16
17 contacts.clear()
18 console.log(contacts.size); // 0
```



MAP - 4

PER CREARE UNA MAPPA SI USA IL COSTRUTTORE «Map», CON I METODI:

- «set» PER AGGIUNGERE UN ELEMENTO;
- «has» PER CONTROLLARE SE L'ELEMENTO ESISTE;
- «get» PER OTTENERE L'ELEMENTO CORRISPONDENTE;
- «delete» PER CANCELLARE L'ELEMENTO;
- «clear» PER SVUOTARE LA MAPPA.

```
1 const contacts : Map<any, any> = new Map();
2 // adds an element
3 contacts.set("Jessie", { phone: "213-555-1234", address: "123 N 1st Ave" });
4 // checks for elements
5 contacts.has( key: "Jessie"); // true
6 contacts.has( key: "Hilary"); // false
7 // retrieves elements
8 contacts.get("Hilary"); // undefined
9 contacts.set("Hilary", { phone: "617-555-4321", address: "321 S 2nd St" });
10 contacts.get("Jessie"); // {phone: "213-555-1234", address: "123 N 1st Ave"}
11 // deletes elements
12 contacts.delete( key: "Raymond"); // false
13 contacts.delete( key: "Jessie"); // true
14 //check size
15 console.log(contacts.size); // 1
16
17 contacts.clear()
18 console.log(contacts.size); // 0
```



MAP - 5

«Map» ESPONE TRE METODI CHE PERMETTONO L'ITERAZIONE DIRETTA:

- «map.keys()» TORNA LE CHIAVI;
- «map.values()» TORNA I VALORI;
- «map.entries()» TORNA GLI ELEMENTI INSERITI SOTTO FORMA DI UN ARRAY [CHIAVE, VALORE]

ESISTE ANCHE UN METODO «.forEach()» SIMILE AGLI ARRAY.

```
1 const map : Map<any, any> = new Map().set(1, 'A').set(2, 'B').set(3, 'C');
2 for (const mapKey of map.keys()) {
3     console.log(mapKey);
4 }
5 // 1
6 // 2
7 // 3
8 for (const mapValue of map.values()) {
9     console.log(mapValue);
10}
11// A
12// B
13// C
14for (const mapEntry : [any, any] of map.entries()) {
15    console.log(mapEntry[0], mapEntry[1]);
16}
17// 1 "A"
18// 2 "B"
19// 3 "C"
20map.forEach( callbackfn: (value, key, map : Map<any, any>) : void => {
21    console.log(`Map has key ${key} and value ${value}, also retrieved with get: ${map.get(key)}`);
22})
23// Map has key 1 and value A, also retrieved with get: A
24// Map has key 2 and value B, also retrieved with get: B
25// Map has key 3 and value C, also retrieved with get: C
```

ESERCIZIO

CONTAPAROLE: SCRIVERE UNO SCRIPT CHE DATA IN INPUT UNA FRASE COMPLESSA «text» RESTITUISCA IN CONSOLE IL NUMERO DI OCCORRENZE DI CIASCUNA PAROLA DELLA FRASE.

USARE «text.split(/\s+/);» PER SPLITTARE UNA STRINGA IN UN'ARRAY DI STRINGHE; ATTENZIONE AL CASING («TEST Test» → «test»: 2).

TEMPO: 20 MINUTI

ESERCIZIO

CREARE UNA FUNZIONE CHE CHIEDA ALL'UTENTE DI INSERIRE UN NOME DI UN PRODOTTO E UNA VALUTAZIONE NUMERICA IN UN UNICO PROMPT SEPARATI DA UN CARATTERE JOLLY A SCELTA.

IL PROMPT CONTINUA AD APPARIRE FINCHÈ L'UTENTE NON PREME «ANNULLA» O CHIUDE O INSERISCE UNA STRINGA SENZA JOLLY.

CONTINUARE ALLA SLIDE SUCCESSIVA.

TEMPO: 15 MINUTI

ESERCIZIO

MEMORIZZARE I DATI CHE L'UTENTE INSERISCE IN UNA MAPPA:
LA CHIAVE DEVE ESSERE IL PRODOTTO, IL VALORE UN OGGETTO CHE
MEMORIZZI:

- TUTTE LE VALUTAZIONI INSERITE (ARRAY DI VALORI);
- LA MEDIA ARITMETICA DELLE VALUTAZIONI.

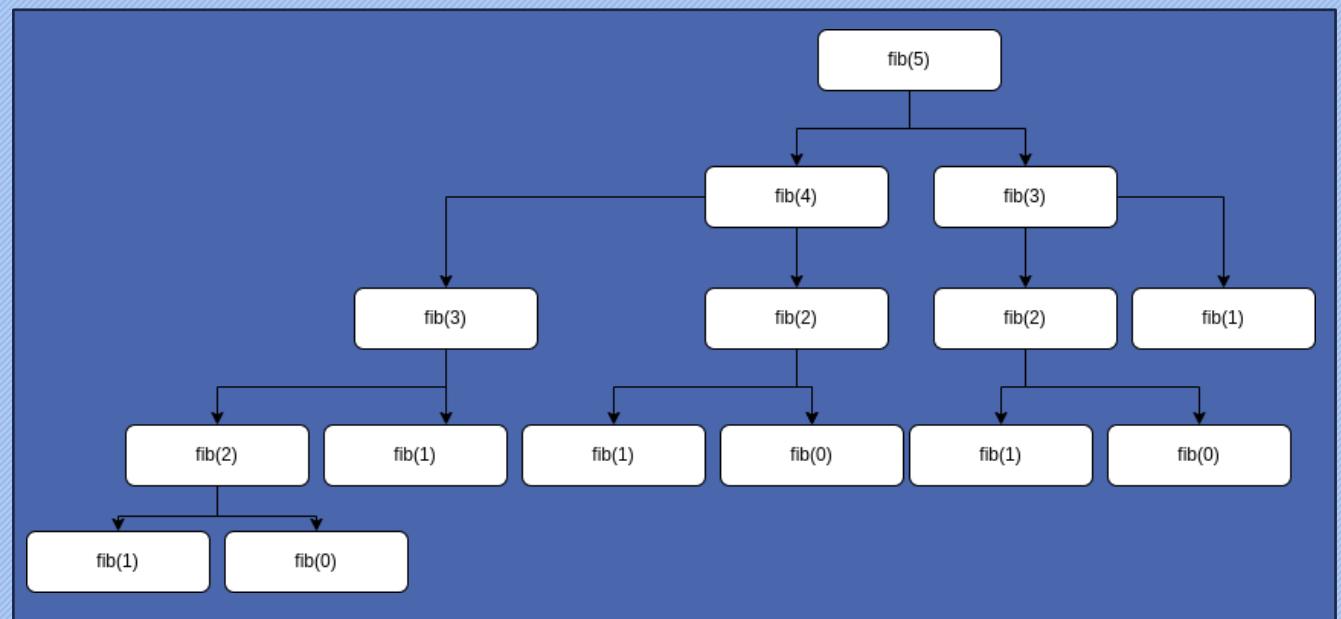
QUANDO L'UTENTE DECIDE DI SMETTERE, LOGGARE TUTTI I DATI
INSERITI.

TEMPO: 25 MINUTI

MAP - 8

RISCRIVERE LA FUNZIONE CHE TORNA LA SEQUENZA DI FIBONACCI SFRUTTANDO LA «MEMOIZATION»:

```
1  function fibonacci(n, memo) { Show usages
2    memo = memo || new Map();
3
4    if (memo.has(n)) {
5      return memo.get(n);
6    }
7
8    if (n <= 1) {
9      return n;
10 } else {
11   return fibonacci(n: n - 1) + fibonacci(n: n - 2);
12 }
13 }
```



SET - 1



UN «Set» IN JAVASCRIPT È UN OGGETTO CHE PERMETTE DI MEMORIZZARE VALORI SENZA CHIAVE.

I VALORI DI UN SET SONO UNIVOCI.

PER CREARE UN SET SI UTILIZZA IL SUO COSTRUTTORE «new Set()».

COME LE MAPPE, SI PUÒ CICLARE DIRETTAMENTE TRAMITE «for...of».

SET - 2

SIMILMENTE ALLE MAPPE, «Set»
ESPONE I METODI:

- «add» PER AGGIUNGERE UN ELEMENTO;
- «has» PER CONTROLLARE SE L'ELEMENTO ESISTE;
- «delete» PER CANCELLARE L'ELEMENTO;
- «clear» PER SVUOTARE IL SET.

NON C'È UN METODO GET.

```
1  let set :Set<any> = new Set();
2
3  let john :{name: string} = { name: "John" };
4  let pete :{name: string} = { name: "Pete" };
5  let mary :{name: string} = { name: "Mary" };
6
7  // visits, some users come multiple times
8  set.add(john);
9  set.add(pete);
10 set.add(mary);
11 set.add(john);
12 set.add(mary);
13
14 // set keeps only unique values
15 console.log(set.size);
16 // 3
17
18 for (let user of set) {
19   console.log(user.name);
20 }
21 // John
22 // Pete
23 // Mary
```



SET - 3

«Set» ESPONE «set.values()»
CHE TORNA I VALORI NEL SET.

ESPONE ANCHE «set.keys()» E
«set.entries()» MA SOLO PER
COMPATIBILITÀ CON LE MAPPE (I
SET NON HANNO CHIAVI).

ANALOGAMENTE ESISTE ANCHE
UN METODO «.forEach()», OVE IL
SECONDO PARAMETRO È UGUALE
AL PRIMO PER COMPATIBILITÀ.

```
1  const set :Set<any> = new Set().add('A').add('B').add('C');
2  for (const setKey of set.keys()) {
3      console.log(setKey);
4  }
5  // A
6  // B
7  // C
8  for (const setValue of set.values()) {
9      console.log(setValue);
10 }
11 // A
12 // B
13 // C
14 for (const setEntry :[any,any] of set.entries()) {
15     console.log(setEntry[0], setEntry[1]);
16 }
17 // A A
18 // B B
19 // C C
20 set.forEach( callbackfn: (value, alwaysValue, set :Set<any>) :void => {
21     console.log(`Set has value ${value}`)
22 })
23 // Set has value A
24 // Set has value B
25 // Set has value C
```

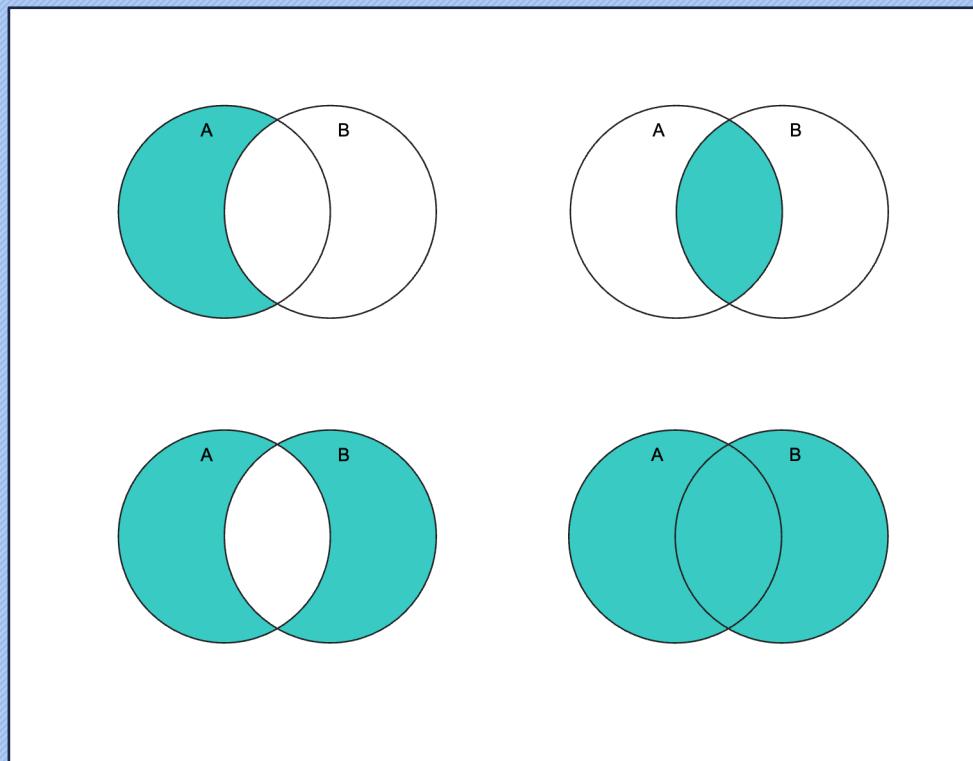
SET - 4

«Set» ESPONE ANCHE METODI
PER COMPORRE DUE SETS:

- «A.difference(B)»;
- «A.intersection(B)»;
- «A.symmetricDifference(B)»;
- «A.union(B)»;

OLTRE AI METODI:

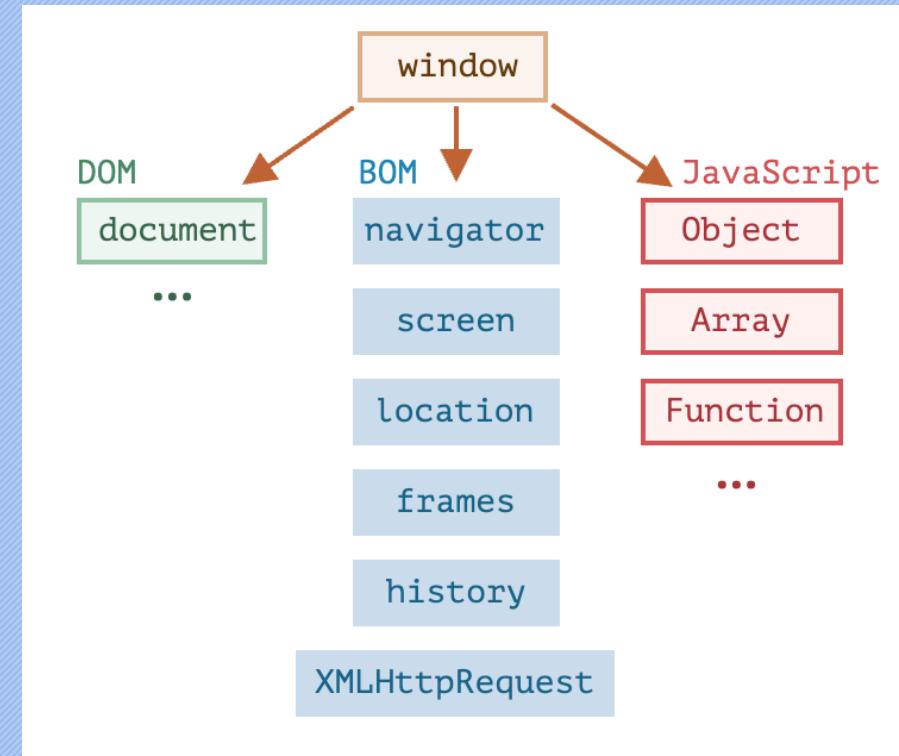
- A.isDisjointFrom(B)
- A.isSubsetOf(B)
- A.isSupersetOf(B)



BROWSER - 1

JAVASCRIPT È STATO CREATO PER ESSERE ESEGUITO IN BROWSER; DA ALLORA SI È EVOLUTO TANTO DA ESSERE POTENZIALMENTE ESEGUIBILE SU QUALSIASI «HOST ENVIRONMENT».

UN «HOST ENVIRONMENT» AGGIUNGE I PROPRI OGGETTI E METODI IN AGGIUNTA AL LINGUAGGIO; NEL BROWSER L'OGGETTO ALLA BASE DI TUTTO È LA «window».

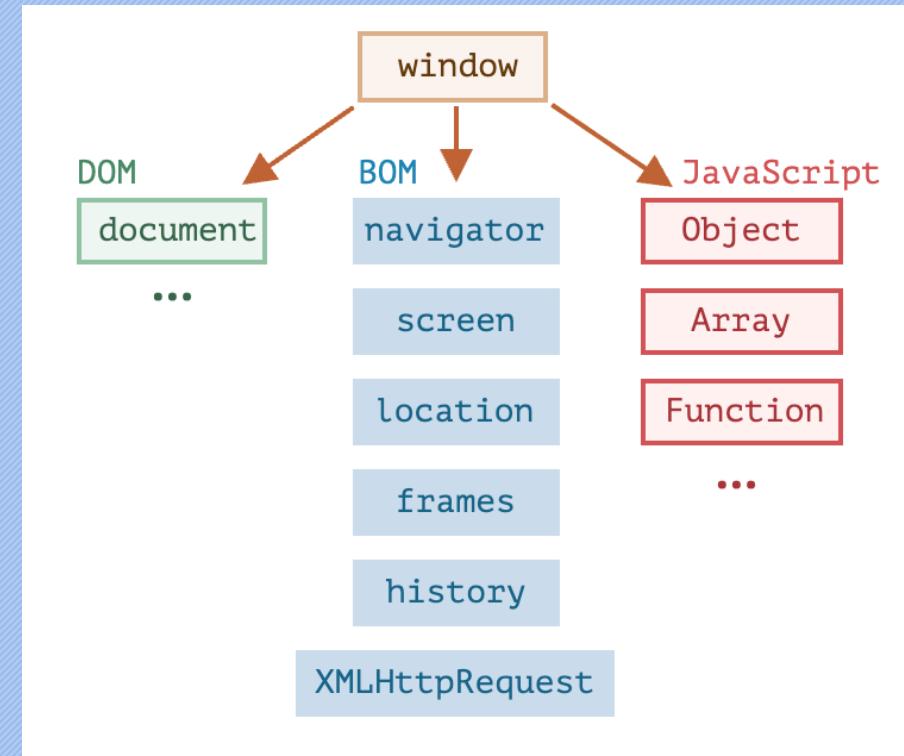


BROWSER - 2

LA «window» È UN OGGETTO GLOBALE CHE RAPPRESENTA LA FINESTRA DEL BROWSER.

IL «DOM» (DOCUMENT OBJECT MODEL) È LO STANDARD PER ACCEDERE E MODIFICARE LA PAGINA WEB. L'«HTML DOM» È UN SUBSET DEL DOM.

IL «BOM» (BROWSER OBJECT MODEL) È L'INSIEME DEGLI OGGETTI AGGIUNTIVI FORNITI DAL BROWSER.

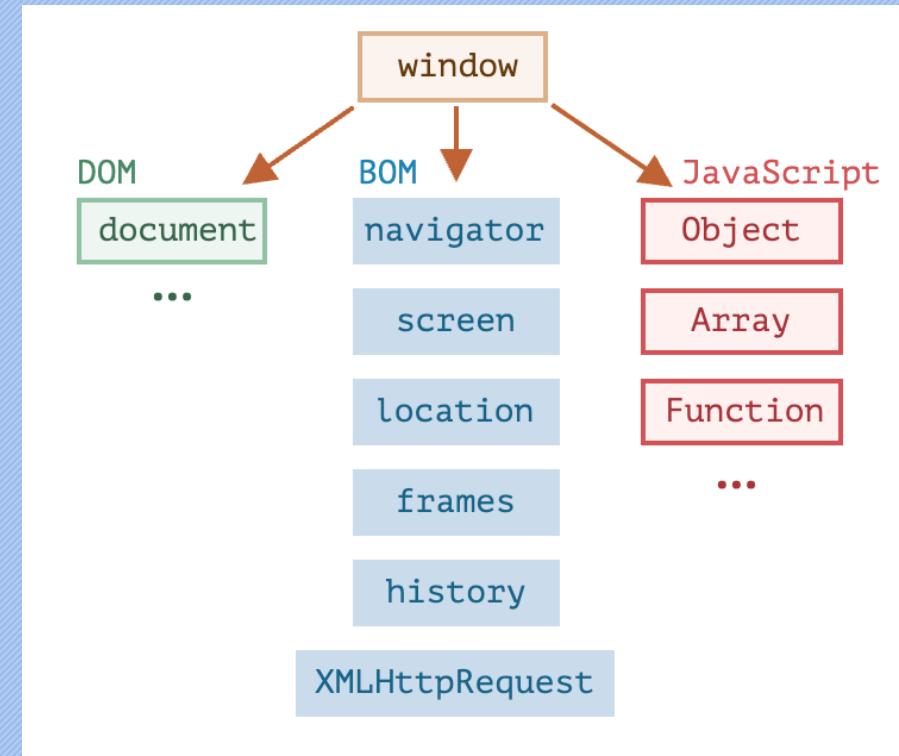


BROWSER - 3

IL «DOM» È UNO STANDARD CODIFICATO DAL W3C CONSORTIUM. È SUDDIVISO IN:

- CORE DOM;
- XML DOM;
- HTML DOM.

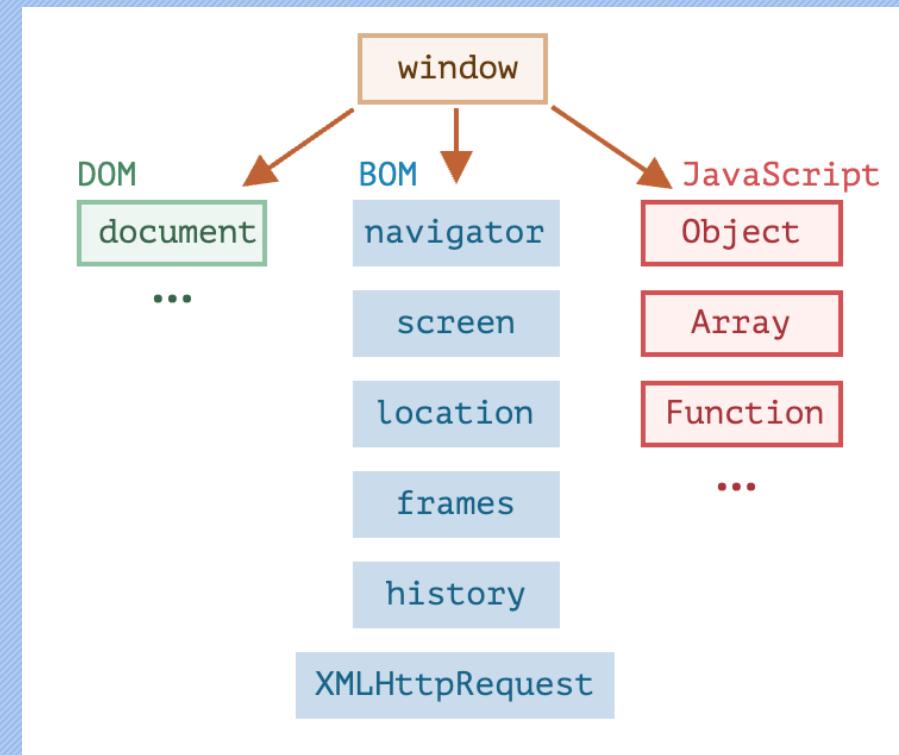
L'«HTML DOM» DEFINISCE PROPERTY, METODI ED EVENTI DI TUTTI GLI ELEMENTI HTML; È LO STANDARD PER AGGUNGERE, CAMBIARE, CANCELLARE, OTTENERE ELEMENTI HTML.



BROWSER - 4

AL CONTRARIO DEL «DOM», IL «BOM» NON È STANDARDIZZATO ANCHE SE È DESCRITTO NELLA SPECIFICA HTML; TUTTI I BROWSER IMPLEMENTANO PIÙ O MENO LE STESSE PROPERTY E METODI.

ESSO PERMETTE L'INTERAZIONE UTENTE CON IL BROWSER, ESCLUSO IL DOCUMENT.



UN DOCUMENTO HTML È COSTITUITO DA TAGS O «HTML ELEMENTS». SECONDO IL DOM, OGNI HTML ELEMENT È UN OGGETTO:

- LE SUE PROPERTY SONO VALORI CHE SI POSSONO LEGGERE / SCRIVERE (ES. PER CAMBIARNE IL CONTENTUTO),
- I SUO METODI SONO AZIONI CHE SI POSSONO SVOLGERE SU DI ESSO (ES. AGGIUNGERE / CANCELLARE).

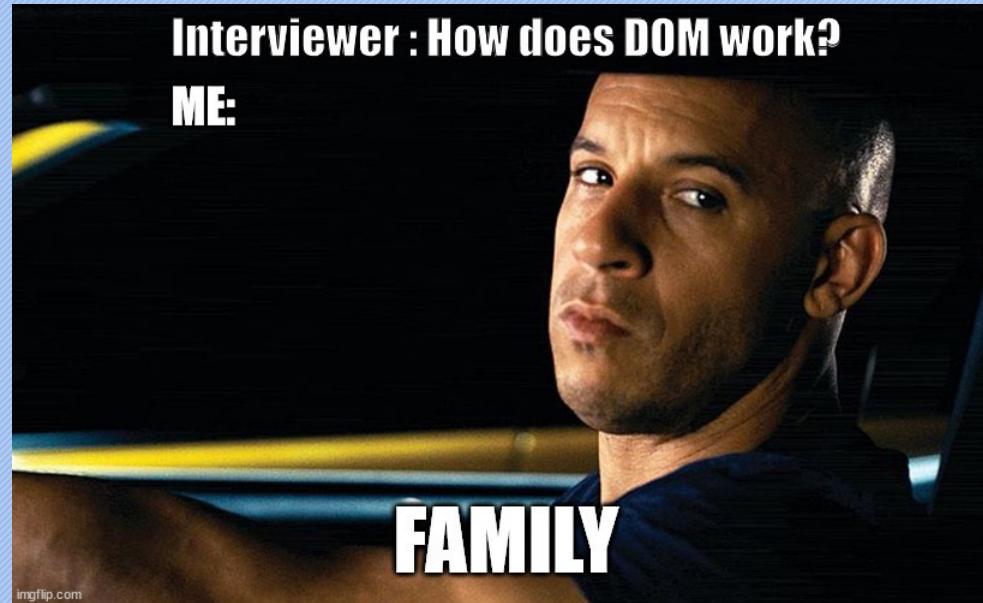
L'OGGETTO DI PARTENZA IN JAVASCRIPT È SEMPRE IL DOCUMENT:

- <html> = document.documentElement
- <body> = document.body
- <head> = document.head

DOM - 2

A PARTIRE DAI TOP TAGS, PER OGNI ELEMENTO SI POSSONO DEFINIRE I SUOI «CHILD NODES», «SIBLINGS» E «DESCENDANTS».

- «CHILD NODES»
TAG CHE SONO «FIGLI DIRETTI»;
SI POSSONO OTTENERE CON
«.childNodes» E, IN PARTICOLARE,
«.firstChild» E «.lastChild»
RESTITUISCONO IL PRIMO E
L'ULTIMO ELEMENTO DELLA LISTA;



- «DESCENDANTS»
TAG CHE SONO INNESTATI NEI «CHILD NODES», QUINDI SONO DISCENDENTI INDIRETTI DEL TAG;
- «SIBLINGS»
TAG CHE SONO FIGLI DELLO STESSO PADRE; SI POSSONO USARE LE PROPERTY «previousSibling» E «nextSibling» PER OTTENERE IL PRECEDENTE E IL SUCCESSIVO, E «parentNode» PER OTTENERE IL PADRE.

TUTTI GLI ELEMENTI OTTENUTI CON LE PROPERTY VISTE SONO DI TIPO «COLLECTIONS»: SONO READ-ONLY, E SI AGGIORNANO AUTOMATICAMENTE SE IL DOM VIENE MODIFICATO.

IL «document» METTE A DISPOSIZIONE ALTRI METODI PER CERCARE TAGS LA CUI POSIZIONE NON È ESATTAMENTE NOTA.

IL PIÙ SEMPLICE È «`getElementById`», CHE TORNA L'ELEMENTO CON L'ID FORNITO, SE PRESENTE.

IMPORTANTE:
USARE SEMPRE ID UNIVOCI!

```
1 // Look for an element with id = myId
2 const elem : HTMLElement = document.getElementById( elementId: 'myId' );
3 if (elem) {
4     // change the background color
5     elem.style.background = 'red';
6 }
```

CON «getElementsByTagName»
SI OTTENGONO INVECE TUTTI I
TAG CON IL TAGNAME FORNITO.

ANALOGAMENTE, SE SI VUOLE
UTILIZZARE COME PARAMETRO DI
RICERCA, RISPETTIVAMENTE, LA
CLASSE CSS O IL NAME, SI
POSSONO USARE I METODI
«getElementsByClassName» E
«getElementsByName».

```
1 //search for all the inputs
2 const elemCollection :HTMLCollectionOf<HTMLInputElement> = document.getElementsByTagName( qualifiedName: 'input');
3
4 //search for all the elements with class "info"
5 const elemClass :HTMLCollectionOf<Element> = document.getElementsByClassName( classNames: 'info');
6
7 //search for all the elements with name equals to "name-attr"
8 const elemName :NodeListOf<HTMLElement> = document.getElementsByName( elementName: 'name-attr');
```

LA RICERCA PUÒ ESSERE RESA PIÙ SPECIFICA UTILIZZANDO UN GENERICO SELETTORE COME PARAMETRO.

SI PUÒ UTILIZZARE INFATTI IL METODO «`.querySelectorAll`» PER CERCARE TUTTI GLI ELEMENTI CHE SODDISFANO IL «CSS SELECTOR» FORNITO.

SE SI VUOLE OTTENERE SOLO IL PRIMO ELEMENTO NEL DOM CHE SODDISFA LA RICERCA, SI PUÒ USARE «`.querySelector`»

SE INVECE SI STA CICLANDO SU UNA COLLECTION E SI VUOLE FILTRARE CONFRONTANDO CON UN SELETTORE, SI PUÒ USARE «`.matches`», CHE RESTITUISCE «`true`» SE L'ELEMENTO CORRISPONDE AL SELETTORE FORNITO.

PER RISALIRE DAI FIGLI AL PRIMO «ANCESTOR TAG» CHE MATCHA UN DATO SELETTORE, SI PUÒ UTILIZZARE IL METODO «`.closest()`», ANCHE QUI PASSANDO UN GENERICO «CSS SELECTOR».

DOM - 8

TABELLA RIASSUNTIVA:

METODO	CERCA PER..	SOLO SUL DOM?	LIVE ELEMENT?
querySelector	CSS-SELECTOR	-	-
querySelectorAll	CSS-SELECTOR	-	-
getElementById	ID	✓	-
getElementsByName	NAME	✓	✓
getElementsByTagName	TAG O '*'	-	✓
getElementsByClassName	CLASSE	-	✓

ESERCIZIO

APRIRE IL QR CODE A LATO E
CERCARE NELLA PAGINA:

- LA «table» CON id="age-table";
- LE «label» NELLA TABELLA;
- IL PRIMO «td» NELLA TABELLA
(«Age»);
- IL «form» CON name="search»;
- IL PRIMO E L'ULTIMO INPUT NEL
«form».



<https://javascript.info/task/find-elements/table.html>

IL DOM ESPONE NUMEROSI METODI PER MODIFICARE L'OGGETTO «document»; È POSSIBILE INFATTI:

- AGGIUNGERE, MODIFICARE O RIMUOVERE GLI ATTRIBUTI DI UN ELEMENTO;
- CAMBIARE LO STILE DI UN ELEMENTO;
- CREARE NUOVI ELEMENTI;
- AGGIUNGERE ELEMENTI IN POSIZIONI SPECIFICHE;
- SOSTITUIRE ELEMENTI CON ALTRI;
- RIMUOVERE ELEMENTI;
- CLONARE ELEMENTI.

QUANDO IL «BROWSER ENGINE» CARICA LA PAGINA, PARSA L'HTML E GENERA GLI OGGETTI DEL DOM.

IN QUESTA FASE, LA MAGGIOR PARTE DEGLI ATTRIBUTI DEI TAG HTML SONO CONVERTITI IN PROPRIETÀ DELL'OGGETTO, IN MODO TALE CHE ESSI SIANO UTILIZZABILI IN JAVASCRIPT.

IL MAPPING NON È 1:1; LA CONVERSIONE VIENE EFFETTUATA SOLO PER GLI ATTRIBUTI «STANDARD» (VEDI MDN) E NON PER QUELLI CREATI DALL'UTENTE.

COME SI PUÒ ACCEDERE A QUESTI ATTRIBUTI?

- «elem.getAttribute»: VERIFICA, DATO IL NOME DELL'ATTRIBUTO, LA SUA ESISTENZA;
- «elem.getAttribute»: RESTITUISCE IL VALORE DELL'ATTRIBUTO;
- «elem.setAttribute»: SETTA UN VALORE PER L'ATTRIBUTO SCELTO;
- «elem.removeAttribute»: RIMUOVE L'ATTRIBUTO.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body id="test" something="non-standard">
8      <script>
9          // standard attribute yields a property
10         alert(document.body.id); // test
11
12         // non-standard attribute does not yield a property...
13         alert(document.body.something); // undefined
14
15         // ...but is accessible via method
16         alert(document.body.getAttribute('something'));// non-standard
17     </script>
18  </body>
19  </html>
```

VA RICORDATO CHE:

- I NOMI DEGLI ATTRIBUTI SONO CASE-SENSITIVE;
- I VALORI SONO SEMPRE DI TIPO STRINGA;
- GLI ATTRIBUTI SI POSSONO ANCHE RECUPERARE DALLA PROPERTY «`attributes`»;
- QUANDO SI SETTA UN ATTRIBUTO, LA PROPRIETÀ CORRISPONDENTE SI SINCRONIZZA CON IL NUOVO VALORE; IL VICEVERSA NON SEMPRE È VERO.

CON JAVASCRIPT SI PUÒ MODIFICARE DIRETTAMENTE SIA LE CLASSI CSS DI UN ELEMENTO SIA IL SUO «style».

È SEMPRE PREFERIBILE OPERARE SULLE CLASSI.

UNO DEI POCHI UTILIZZI ACCETTABILI DI MODIFICA DIRETTA ALLO «style» È AD ES. IL CALCOLO DINAMICO DELLA POSIZIONE DI UN ELEMENTO (UTILIZZANDO «elem.style.top» / «elem.style.left»).

POICHÉ «class» È UNA RESERVED WORD DI JAVASCRIPT, LA PROPERTY CORRISPONDENTE È CHIAMATA «className».

MODIFICARE «className» MODIFICA L'INTERO VALORE DELL'ATTRIBUTO «class».

SE INVECE SI VUOLE OPERARE SU UNA SINGOLA CLASSE, ESISTE LA PROPERTY «classList», CHE ESPONE I METODI:

- «add/remove», CHE AGGIUNGE/RIMUOVE LA CLASSE;
- «toggle», CHE AGGIUNGE LA CLASSE SE NON C'È ALTRIMENTI LA RIMUOVE;
- «contains», CHE TORNA true SE «classList» CONTIENE LA CLASSE DATA.

DOM - 16

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body class="main">
8  <script>
9      document.body.className = "my-custom-new-class";
10     alert(document.body.className); // my-custom-new-class
11
12     document.body.classList.add('another-class');
13     alert(document.body.className); // my-custom-new-class another-class
14
15     document.body.classList.toggle('my-custom-new-class');
16     alert(document.body.className); // another-class
17
18     if (document.body.classList.contains('another-class')) {
19         document.body.classList.remove('another-class');
20     }
21     alert(document.body.className); // ""
22 </script>
23 </body>
24 </html>
```

LA PROPERTY «style» DI UN ELEMENTO RISPECCHIA QUANTO INSERITO NELL'ATTRIBUTO OMONIMO. LE SUE PROPRIETÀ MANTENGONO GLI STESSI NOMI; QUELLE CON NOMI COMPOSTI PASSANO IN CAMEL-CASE.

IL RESET DI UNA PROPERTY SI FA RIASSEGNANDO A STRINGA VUOTA O VIA «removeProperty».

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8  <script>
9      document.body.style.width = "100px";
10     document.body.style.height = "100px";
11     document.body.style.backgroundColor = 'blue';
12     document.body.style.display = "none";
13
14     setTimeout(() => document.body.style.display = "", 1000);
15     setTimeout(() => document.body.style.removeProperty('background'), 1000);
16  </script>
17  </body>
18  </html>
```

QUANDO SI RIASSEGNA UNA PROPRIETÀ DI STILE, BISOGNA SEMPRE RICORDARE DI AGGIUNGERE L'UNITÀ DI MISURA, SE NECESSARIA.

QUANTO VISTO FINORA VALE IN SCRITTURA; SE PROVASSIMO A LEGGERE IN QUESTO MODO I VALORI DI «style», OTTERREMMO QUASI SEMPRE STRINGA VUOTA.

IN LETTURA INFATTI SI USA IL METODO «getComputedStyle(elem)», CHE RESTITUISCE UN OGGETTO CON TUTTE LE PROPRIETÀ DELLO STILE CON I VALORI RISOLTI E NON CALCOLATI (MALGRADO IL NOME).

DOM - 19

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6      <style> body { color: red; margin: 5px } </style>
7  </head>
8  <body>
9
10 <script>
11     let computedStyle = getComputedStyle(document.body);
12
13     // now we can read the margin and the color from it
14     alert( computedStyle.marginTop ); // 5px
15     alert( computedStyle.color ); // rgb(255, 0, 0)
16 </script>
17
18 </body>
19 </html>
```

PER AGGIUNGERE ELEMENTI AL DOM VIA JAVASCRIPT SONO NECESSARI DUE PASSAGGI DISTINTI: PRIMA SI DEVONO CREARE GLI ELEMENTI, SETTANDO TUTTE LE PROPERTY E GLI ATTRIBUTI NECESSARI, QUINDI ESSI DEVONO ESSERE «AGGANCIATI» AL DOM.

PER LA CREAZIONE, IL «document» ESPONE DUE METODI:

- «createElement» CREA UN ELEMENTO HTML CON IL TAG FORNITO;
- «createTextNode» CREA UN ELEMENTO DI TESTO COL TESTO FORNITO.

DOM - 21

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8  <script>
9      // 1. Create <div> element
10     let div = document.createElement('div');
11
12     // 2. Set its class to "alert"
13     div.className = "alert";
14
15     // 3. Fill it with the content
16     div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";
17
18     // 4. Add attribute
19     div.setAttribute('my-attr', 'true');
20
21     // Inspect the DOM: where's the element?
22 </script>
23 </body>
24 </html>
```

PER MOSTRARE L'ELEMENTO «div» CHE ABBIAMO CREATO DOBBIAMO INSERIRLO NEL «document». PER FARLO ABBIAMO A DISPOSIZIONE:

- IL METODO «append», CHE AGGIUNGE AL PADRE L'ELEMENTO FORNITO COME ULTIMO NODO.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8      <div id="container">
9          <div>First element</div>
10     </div>
11     <script>
12         let div = document.createElement('div');
13         div.className = "alert";
14         div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";
15         div.setAttribute('my-attr', 'true');
16
17         const container = document.querySelector('#container');
18         container.append(div)
19
20     </script>
21  </body>
22  </html>
23  <body>
24
25  <!--
26  <div id="container">
27      <div> First element </div>
28      <div class="alert" my-attr="true">
29          <strong>Hi there!</strong> You've read an important message.
30      </div>
31  </div>
32  <!-->
```

PER MOSTRARE L'ELEMENTO «div» CHE ABBIAMO CREATO DOBBIAMO INSERIRLO NEL «document». PER FARLO ABBIAMO A DISPOSIZIONE:

- IL METODO «prepend», CHE AGGIUNGE L'ELEMENTO FORNITO AL PADRE COME PRIMO NODO.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8  <div id="container">
9      <div>First element</div>
10 </div>
11 <script>
12     let div = document.createElement('div');
13     div.className = "alert";
14     div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";
15     div.setAttribute('my-attr', 'true');

16
17     const container = document.querySelector('#container');
18     container.prepend(div)
19 </script>
20 </body>
21 </html>
22 <body>
23
24 <!--
25 <div id="container">
26     <div class="alert" my-attr="true">
27         <strong>Hi there!</strong> You've read an important message.
28     </div>
29     <div> First element </div>
30 </div>
31 <!-->
```

PER MOSTRARE L'ELEMENTO «div» CHE ABBIAMO CREATO DOBBIAMO INSERIRLO NEL «document». PER FARLO ABBIAMO A DISPOSIZIONE:

- IL METODO «before», CHE AGGIUNGE L'ELEMENTO PRIMA DELL'ELEMENTO SPECIFICATO.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8      <div id="container">
9          <div>First element</div>
10     </div>
11     <script>
12         let div = document.createElement('div');
13         div.className = "alert";
14         div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";
15         div.setAttribute('my-attr', 'true');
16
17         const container = document.querySelector('#container');
18         container.before(div)
19     </script>
20     </body>
21     </html>
22     <body>
23
24     <!--
25     <div class="alert" my-attr="true">
26         <strong>Hi there!</strong> You've read an important message.
27     </div>
28     <div id="container">
29         <div> First element </div>
30     </div>
31     <-->
```

PER MOSTRARE L'ELEMENTO «div» CHE ABBIAMO CREATO DOBBIAMO INSERIRLO NEL «document». PER FARLO ABBIAMO A DISPOSIZIONE:

- IL METODO «`after`», CHE AGGIUNGE L'ELEMENTO DOPO L'ELEMENTO SPECIFICATO.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8  <div id="container">
9      <div>First element</div>
10 </div>
11 <script>
12     let div = document.createElement('div');
13     div.className = "alert";
14     div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";
15     div.setAttribute('my-attr', 'true');
16
17     const container = document.querySelector('#container');
18     container.append(div)
19 </script>
20 </body>
21 </html>
22 <body>
23
24 <!--
25 <div id="container">
26     <div> First element </div>
27 </div>
28 <div class="alert" my-attr="true">
29     <strong>Hi there!</strong> You've read an important message.
30 </div>
31 <-->
```

QUESTI QUATTRO METODI POSSONO ANCHE ESSERE USATI PER AGGIUNGERE DEL TESTO, PASSANDO UNA STRINGA COME ARGOMENTO, AD ES. «document.body.append('Hello')».

TUTTAVIA, SE LA STRINGA CONTIENE DELL'HTML, QUESTO NON VIENE RENDERIZZATO MA VIENE AGGIUNTO COME TESTO.

SE SI VUOLE AGGIUNGERE AL «document» UNA STRINGA CHE CONTIENE HTML IN MANIERA ANALOGA ALLA PROPERTY «innerHTML» SI PUÒ UTILIZZARE IL METODO «insertAdjacentHTML».

IL METODO RICHIENDE DUE ARGOMENTI, IL SECONDO È LA STRINGA DA INSERIRE, IL PRIMO È LA POSIZIONE:

- "beforebegin"
- "afterbegin"
- "beforeend"
- "afterend"

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8  <div id="container">
9      <div>First element</div>
10 </div>
11 <script>
12     const container = document.querySelector('#container');
13     container.insertAdjacentHTML('beforebegin', '<p>beforebegin</p>');
14     container.insertAdjacentHTML('afterbegin', '<p>afterbegin</p>');
15     container.insertAdjacentHTML('beforeend', '<p>beforeend</p>');
16     container.insertAdjacentHTML('afterend', '<p>afterend</p>');
17 </script>
18 </body>
19 </html>
20 <body>
21
22 <!--
23 <p>beforebegin</p>
24 <div id="container">
25     <p>afterbegin</p>
26     <div>First element</div>
27     <p>beforeend</p>
28 </div>
29 <p>afterend</p>
30 <!-->
```

PER RIMUOVERE UN NODO L'UNICO
METODO DISPONIBILE È IL
«remove».

SE USIAMO «append» / «prepend» /
ECC SU UN NODO ESISTENTE, ESSO
VIENE SPOSTATO, NON C'È BISOGNO
DI CANCELLARLO.

SE INVECE VOGLIAMO SOSTITUIRE
UN NODO CON UN ALTRO, SI PUO
USARE «replaceWith».

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8  <div>
9      <p id="first">First element</p>
10     <p id="second">Second element</p>
11  </div>
12  <script>
13      document.querySelector('#first').remove();
14      document.querySelector('#second').replaceWith('Hello');
15  </script>
16  </body>
17  </html>
18  <body>
19
20  <!--
21  <div>
22      Hello
23  </div>
24  -->
```

CON IL METODO «`cloneNode`»
SI PUÒ CREARE UNA COPIA DI UN
NODO ESISTENTE.

PASSANDO «`true`» COME
PARAMETRO SI CLONANO ANCHE
TUTTI GLI ELEMENTI INTERNI.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8      <div>
9          <p id="first">First element</p>
10     </div>
11     <script>
12         const first = document.querySelector('#first');
13         const second = first.cloneNode(true);
14         second.id = 'second';
15         second.innerHTML = 'Second element';
16         first.append(second);
17     </script>
18     </body>
19     </html>
20     <body>
21
22     <!--
23     <p id="first">
24         First element
25         <p id="second">
26             Second element
27         </p>
28     </p>
29     -->
```

ESERCIZIO:

PARTENDO DALL'HTML A LATO,
CREARE LO SCRIPT E
AGGIUNGERE LE DUE FUNZIONI
MANCANTI CHE AGGIUNGONO
RIGHE E COLONNE ALLA
TABELLA.

TEMPO: 20 MINUTI

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset=utf-8 />
5      <title>Table</title>
6      <script src="table.js"></script>
7  </head>
8  <body>
9      <div class="buttons">
10         <input type="button" onclick="insertRow()" value="Insert row">
11         <input type="button" onclick="insertColumn()" value="Insert Column">
12     </div>
13     <table id="sampleTable">
14         <thead id="tHead">
15             <tr id="rowHead">
16                 <th>Header 1</th>
17                 <th>Header 2</th>
18             </tr>
19         </thead>
20         <tbody id="tBody">
21             <tr>
22                 <td>Row 1 cell 1</td>
23                 <td>Row 1 cell 2</td>
24             </tr>
25             <tr>
26                 <td>Row 2 cell 1</td>
27                 <td>Row 2 cell 2</td>
28             </tr>
29         </tbody>
30     </table>
31 </body>
32 </html>
```

ESERCIZIO: AGGIUNGERE:

- UN FORM CON TRE INPUT:
INDICE RIGA, INDICE COLONNA
E VALORE
- UN PULSANTE CHE AGGIORNA
LA CELLA INDIVIDUATA DAI DUE
INDICI COL VALORE FORNITO.

TEMPO: 20 MINUTI

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset=utf-8 />
5      <title>Table</title>
6      <script src="table.js"></script>
7  </head>
8  <body>
9      <div class="buttons">
10         <input type="button" onclick="insertRow()" value="Insert row">
11         <input type="button" onclick="insertColumn()" value="Insert Column">
12     </div>
13     <table id="sampleTable">
14         <thead id="tHead">
15             <tr id="rowHead">
16                 <th>Header 1</th>
17                 <th>Header 2</th>
18             </tr>
19         </thead>
20         <tbody id="tBody">
21             <tr>
22                 <td>Row 1 cell 1</td>
23                 <td>Row 1 cell 2</td>
24             </tr>
25             <tr>
26                 <td>Row 2 cell 1</td>
27                 <td>Row 2 cell 2</td>
28             </tr>
29         </tbody>
30     </table>
31 </body>
32 </html>
```

EVENTS - 1



COME DEFINIZIONE GENERALE, POSSIAMO DIRE CHE UN «EVENTO» È UN SEGNALE CHE QUALCOSA È ACCADUTO.

ESEMPI DI EVENTI SONO IL CLICK O L'HOVER DEL MOUSE, LA PRESSIONE DI UN TASTO SULLA TASTIERA, IL SUBMIT DI UN FORM ECC.

SI POSSONO ASSEGNAME FUNZIONI («EVENT HANDLERS») DA ESEGUIRE QUANDO SI CATTURA UN DETERMINATO TIPO DI EVENTO. GLI HANDLERS SI AGGIUNGONO AGLI ELEMENTI HTML TRAMITE GLI ATTRIBUTI «on<event>» CORRISPONDENTE ALL'EVENTO IN QUESTIONE («click» → »onclick», «keydown» → «onkeydown»...).

EVENTS - 2

AD ESEMPIO, I DUE METODI CREATI DELL'ESERCIZIO PRECEDENTI SONO AGGANCIATI AI DUE PULSANTI TRAMITE «`onclick`».

IN ALTERNATIVA AL BINDING NELL'HTML, SI PUÒ USARE IN JAVASCRIPT LA PROPERTY CORRISPONDENTE (`«button.onclick = ...»`).

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset=utf-8 />
5      <title>Table</title>
6      <script src="table.js"></script>
7  </head>
8  <body>
9      <div class="buttons">
10         <input type="button" onclick="insertRow()" value="Insert row">
11         <input type="button" onclick="insertColumn()" value="Insert Column">
12     </div>
13     <table id="sampleTable">
14         <thead id="tHead">
15             <tr id="rowHead">
16                 <th>Header 1</th>
17                 <th>Header 2</th>
18             </tr>
19         </thead>
20         <tbody id="tBody">
21             <tr>
22                 <td>Row 1 cell 1</td>
23                 <td>Row 1 cell 2</td>
24             </tr>
25             <tr>
26                 <td>Row 2 cell 1</td>
27                 <td>Row 2 cell 2</td>
28             </tr>
29         </tbody>
30     </table>
31 </body>
32 </html>
```

EVENTS - 3

QUESTO APPROCCIO HA IL VANTAGGIO DI ESSERE SINTETICO E IMMEDIATO, MA HA UN PROBLEMA:
SI PUÒ ASSEGNAME UN SOLO HANDLER A UN SOLO ELEMENTO HTML.

SI PUÒ OVVIARE USANDO I METODI «`addEventListener`» / «`removeEventListener`».

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Demo</title>
6  </head>
7  <body>
8      <div>
9          <input type="button" value="Click me" id="one"/>
10     </div>
11     <script>
12         document.getElementById('one').addEventListener('click', (e) => console.log(e.target.id));
13         document.getElementById('one').addEventListener('click', (e) => e.target.value = 'Clicked!');
14     </script>
15     </body>
16     </html>
17     </body>
```



IN QUESTO MODO SI POSSONO AGGIUNGERE ALLO STESSO ELEMENTO PIÙ FUNZIONI DA ESEGUIRE SULLO STESSO EVENTO.

COME TERZO PARAMETRO DI QUESTE DUE FUNZIONI SI PUÒ AGGIUNGERE UN OGGETTO CHE AGGIUNGE OPZIONI SPECIFICHE:

- «once»: SE «true» RIMUOVE IL LISTENER DOPO LA PRIMA ESECUZIONE;
- «capture»: GESTISCE LA FASE DI CATTURA DELL'EVENTO (VEDREMO IN SEGUITO);
- «passive»: SEGNALA AL BROWSER CHE L'HANDLER NON CHIAMA «preventDefault» (VEDREMO IN SEGUITO).

EVENTS - 5



PER RIMUOVERE UN EVENT LISTENER SI USA «removeEventListener».

BISOGNA FARE ATTENZIONE A PASSARE LA STESSA FUNZIONE AI DUE METODI DI AGGIUNTA E DI RIMOZIONE, ALTRIMENTI LA RIMOZIONE NON AVVIENE;

SE «once» NON È «true» ALLORA È NECESSARIO SALVARE LA FUNZIONE UTILIZZATA PRIMA DI PASSARLA NELL' «addEventListener» COSÌ DA POTERLA POI RIPASSARE AL «removeEventListener».

PER OGNI EVENTO EMESSO, IL BROWSER CREA UN OGGETTO EVENTO A CUI SI PUÒ ACCEDERE PER AVERE INFORMAZIONI SU COSA È AVVENUTO.

NELLO SPECIFICO, SI PUÒ ACCEDERE A:

- «`type`», OVVERO IL TIPO DELL'EVENTO;
- «`currentTarget`», OVVERO L'ELEMENTO HTML CHE STA ESEGUENDO LA FUNZIONE HANDLER;
- «`clientX`» E «`clientY`», OVVERO LE COORDINATE NELLA FINESTRA DELL'EVENTO;
- ...

EVENTS - 7



ESERCIZIO

IMPLEMENTARE IL GIOCO DEL TRIS PER DUE GIOCATORI:

- CREARE UNA BOARD 3x3;
- AGGIUNGERE UN LISTENER SU CIASCUNA DELLE NOVE CASELLE CHE INSERISCE «x» OPPURE «o» A SECONDA DEL GIOCATORE;
- AD OGNI CLICK VERIFICARE SE IL GIOCO È TERMINATO E MOSTRARE IL VINCITORE.

TEMPO: 30 MINUTI

DUE CONCETTI CARDINE RIGUARDO AGLI EVENTI SONO IL «*bubbling*» E IL «*capturing*».

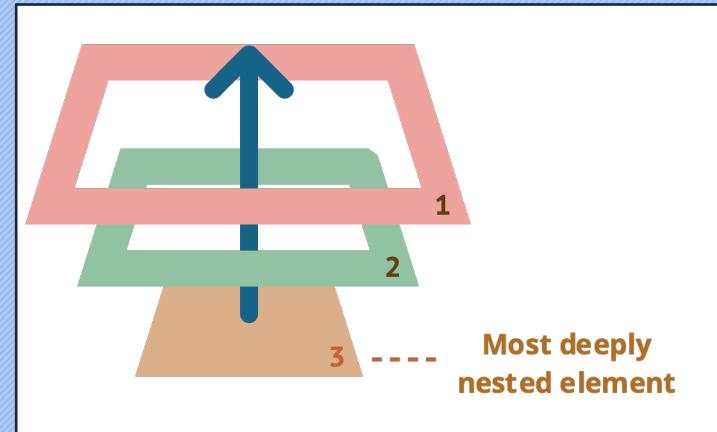
GENERANDO UN EVENTO SU UN ELEMENTO, ESSO VIENE CATTURATO DALL'«*EVENT HANDLER* SULL'ELEMENTO (SE PRESENTE), QUINDI SUL PADRE, QUINDI SU TUTTI GLI «*ancestor*» FINO AL «*document*».

QUESTO PROCESSO DAL BASSO VERSO L'ALTO È DETTO «*bubbling*».

QUASI TUTTI GLI EVENTI SON INTERESSATI (NON LO È IL «*focus*»).

EVENTS - 9

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <title>Title</title>
6  </head>
7  <body>
8
9  <form onclick="alert('form')">
10   FORM
11   <div onclick="alert('div')">
12     DIV
13     <p onclick="alert('p')">P</p>
14   </div>
15 </form>
16 <!--
17 p
18 div
19 form
20 -->
21
22 </body>
23 </html>
```



L'ELEMENTO PIÙ INNESTATO CHE HA CAUSATO L'EVENTO PUÒ ESSERE ACCESSIBILE TRAMITE LA PROPRIETÀ «target» DELL'EVENTO.

C'È DIFFERENZA CON «currentTarget»; ESSO INFATTO INDICA L'ELEMENTO CHE STA ATTUALMENTE LANCIANDO LA FUNZIONE DI HANDLER.

NELL'ESEMPIO PRECEDENTE, CLICCANDO SUL «p», ESSO È IL «target» MENTRE A SECONDA DI DOVE È ARRIVATO L'EVENTO DURANTE IL «bubbling» AVREMO UN DIVERSO «currentTarget».

PER STOPPARE IL «bubbling» È POSSIBILE UTILIZZARE IL METODO «stopPropagation» DELL'OGGETTO EVENTO.

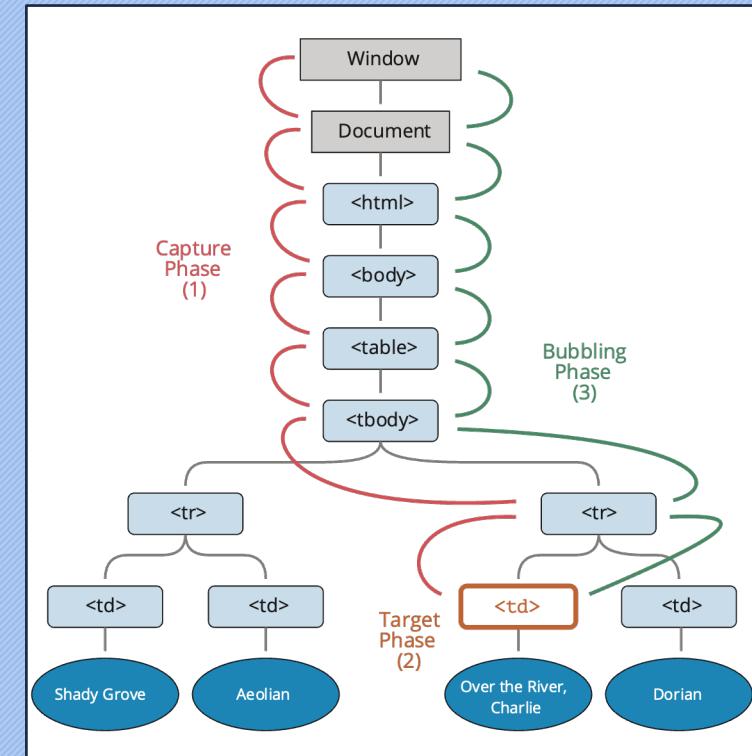
SE CI SONO PIÙ HANDLERS SULLO STESSO ELEMENTO E IL METODO «stopPropagation» È CHIAMATO SOLO SU UNO, I SUCCESSIVI ESEGUIRANNO COMUNQUE LA FUNZIONE ASSEGNATA.

SE SI VUOLE STOPPARE IMMEDIATAMENTE ANCHE LE ALTRE FUNZIONI, SI PUÒ USARE «stopImmediatePropagation»

EVENTS - 12

IL «capturing» È L' ‘OPPOSTO’
DEL «bubbling».

LA FASE DI «bubbling» È SOLO LA
TERZA DELLA PROPAGAZIONE
DELL'EVENTO: PRIMA DI ESSE CI
SONO APPUNTO IL «capturing»,
IN CUI L'EVENTO SI PROPAGA
DALLA «Window» IN GIÙ, QUINDI
LA FASE DI TARGET, IN CUI
L'EVENTO RAGGIUNGE
L'ELEMENTO SELEZIONATO.



IL «capturing» È RARAMENTE USATO, TUTTAVIA A VOLTE PUÒ ESSERE UTILE.

PER FAR SI CHE UN «EVENT HANDLER» SIA ESEGUITO IN FASE DI «capturing» SI PUÒ PASSARE «{capture: true}» COME «option» QUANDO SI AGGANCIA IL LISTENER ALL'ELEMENTO.

SI PUÒ TRARRE VANTAGGIO DEI DUE CONCETTI SOPRA ESPOSTI PER IMPLEMENTARE IL PATTERN CHIAMATO «EVENT DELEGATION».

IN SINTESI, SE CI SONO TANTI ELEMENTI DELLO STESSO TIPO E VOGLIAMO ASCOLTARE UN EVENTO (CHE AMMETTE IL BUBBLING) SU DI ESSI, DOVREMMO ASSEGNARE UN HANDLER A CIASCUNO.

SI PUÒ SEMPLIFICARE ASSEGNAndo UN UNICO HANDLER ALL'ELEMENTO PADRE COMUNE E USANDO «event.target» PER CAPIRE QUALE ELEMENTO HA LANCIATO L'EVENTO.

EVENTS - 15



ESERCIZIO

VERIFICARE SE È POSSIBILE USARE IL PATTERN DI «EVENT DELEGATION» SULL'ESERCIZIO PRECEDENTE.

TEMPO: 15 MINUTI

GLI ELEMENTI NATIVI DEL BROWSER HANNO UN COMPORTAMENTO DI DEFAULT:

- UN CLICK SU LINK NAVIGA ALL' «`href`» SPECIFICATO
- UN CLICK SU UN «`button type='submit'`» INVIA I DATI AL SERVER
- ...

PUÒ CAPITARE CHE SI VOGLIA SOVRASCRIVERE IL COMPORTAMENTO DI DEFAULT PER AGGIUNGERE UN «`EVENT HANDLER`» CUSTOM.

SI OTTIENE QUESTO RISULTATO IN DUE MODI:

- CHIAMANDO «`preventDefault()`» SULL'OGGETTO EVENTO NELL'HANDLER;
- RESTITUENDO «`false`» ALLA FINE DELL'HANDLER.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8      <a href="#" onclick="e.preventDefault()">
9          This one won't navigate.
10     </a>
11     <a href="#" onclick="return false">
12         Neither this one.
13     </a>
14  </body>
15  </html>
```

QUANDO SI AGGANCIA IL LISTENER ALL'ELEMENTO SI PUÒ PASSARE «{passive: true}» COME «option» PER SEGNALARE AL BROWSER CHE L'EVENTO DI DEFAULT NON SARÀ BLOCCATO.

QUESTO PUÒ ESSERE UTILE AD ES. PER EVENTI SU MOBILE COME IL «touchmove», CHE GENERANO, SE NON PREVENUTI, UN FLUSSO DI EVENTI DI SCROLLING;

IL BROWSER DEVE PRIMA PROCESSARE TUTTI GLI EVENTI DI QUESTO TIPO E SE L'EVENTO NON È BLOCCATO, ALLORA SCROLLARE.

CON «passive: true», LO SCROLLING È IMMEDIATO.

EVENTI LANCIATI TRAMITE MOUSE:

- «mousedown» / «mouseup»: TASTO SINISTRO DEL MOUSE PREMUTO/RILASCIATO SU UN ELEMENTO;
- «mouseover» / «mouseout»: PUNTATORE DEL MOUSE CHE ENTRA/ESCE DA UN ELEMENTO;
- «mousemove»: OGNI MOVIMENTO DEL MOUSE;
- «click»: LANCIATO DOPO «mousedown» / »mouseup» SULLO STESSO ELEMENTO;
- «dblclick»: LANCIATO DOPO DUE CLICK CONSECUTIVI SULLO STESSO ELEMENTO;
- contextmenu: TASTO DESTRO DEL MOUSE PREMUTO SU UN ELEMENTO.
- ...

ESERCIZIO

IMPLEMENTARE UN SEMPLICE «DRAG & DROP».

- CREARE UN ELEMENTO (ES. DIV QUADRATO);
- AL «mousedown» MEMORIZZARE LA POSIZIONE INIZIALE (X / Y);
- AL «mousemove» SPOSTARE L'ELEMENTO MODIFICANDONE LA POSIZIONE RISPETTO A QUANTO SALVATO (MODIFICARE «el.style»);
- RICORDARE DI IMPLEMENTARE ANCHE «mouseup» E «dragstart» (SI, CI SONO EVENTI NATIVI SPECIFICI PER IL «drag» MA VOGLIAMO DISATTIVARLI; COME?)

TEMPO: 15 MINUTI.

EVENTI LANCIATI TRAMITE TASTIERA:

- «keydown»: EVENTO LANCIATO QUANDO UN TASTO VIENE PREMUTO;
- «keyup»: EVENTO LANCIATO QUANDO UN TASTO VIENE RILASCIATO;
- «keypress»: OBSOLETO, DA NON USARE.

LE INFORMAZIONI SU QUALE TASTO È STATO PREMUTO POSSONO ESSERE RICAVATE DALLE PROPERTY «key» / «code» SULL'OGGETTO EVENTO: LA PRIMA CORRISPONDE ALLA LETTERA, LA SECONDA INVECE AL TASTO FISICO PREMUTO.

CHE DIFFERENZA C'È TRA «key» E «code»?

- «key» DIPENDE DALLA LOCALIZZAZIONE DELLA TASTIERA:
AD ESEMPIO LA TASTIERA TEDESCA HA LA «z» AL POSTO DELLA «y»,
QUINDI PREMENDO «y» AVREMMO «key = y» E «code = KeyZ».
- «key» CAMBIA ANCHE TRA MAIUSCOLE E MINUSCOLE:

	«key»	«code»
Shift + z = «Z»	«Z»	«KeyZ»
«z»	«z»	«KeyZ»

I TASTI SPECIALI HANNO VALORI BEN DEFINITI, TRA CUI:

	«key»	«value»
INVIO	«Enter»	«Enter»
SPAZIO	« »	«Space»
FRECCE	«Arrow<direzione>»	«Arrow<direzione>»
CTRL/CONTROL	«Control»	«ControlLeft/Right»
ALT/OPTION	«Alt»	«AltLeft/Right»
WIN/COMMAND	«Meta»	«MetaLeft/Right»
...

PER RICONOSCERE LE COMBINAZIONI DI TASTI (SHIFT, CTRL,...) SI POSSONO USARE LE PROPERTY DELL'OGGETTO EVENTO:

- «altKey»: «true» SE È STATO PREMUTO «Alt»/«Option»;
- «ctrlKey»: «true» SE È STATO PREMUTO «Ctrl»/«Control»;
- «metaKey»: «true» SE È STATO PREMUTO «Win»/«Command»;
- «shiftKey»: «true» SE È STATO PREMUTO «Shift».

LINK UTILI:

<https://www.w3.org/TR/uievents-key/>

<https://www.w3.org/TR/uievents-code/>

ESERCIZIO

RIPRENDERE L'ESERCIZIO PRECEDENTE:

- QUANDO SI PREMONO LE FRECCE L'ELEMENTO DEVE SPOSTARSI DI UNA DISTANZA FISSATA NELLA DIREZIONE CORRISPONDENTE;
- QUANDO SI PREME INVIO L'ELEMENTO DEVE RUOTARE IN SENSO ORARIO, QUANDO SI PREME BACKSPACE IN SENSO ANTIORARIO;
- QUANDO SI PREME ESC L'ELEMENTO DEVE SPARIRE/RIAPPARIRE.

È AMMESSO MODIFICARE «el.style»; GRADITE ALTRE VARIAZIONI.

TEMPO: 15 MINUTI

ESERCIZIO

A PARTIRE DA UNA LISTA DI ELEMENTI ‘``’/ ‘``’:

- VISUALIZZARE L'INGRESSO DEL MOUSE SU UN ELEMENTO (ES. CAMBIANDO IL COLORE DI SFONDO) E RESETTARE IL COMPORTAMENTO ALL'USCITA;
- VISUALIZZARE LA SELEZIONE DI UN ELEMENTO TRAMITE CLICK (ES. CAMBIANDO IL COLORE DI SFONDO);
- IL SECONDO PUNTO HA PRECEDENZA SUL PRIMO;
- SE SI TIENE PREMUTO «CTRL» (O «COMMAND» SU MAC) SI POSSONO SELEZIONARE PIÙ ELEMENTI.

TEMPO: 20 MINUTI.

ESERCIZIO

CREARE UNA SELECT CUSTOMIZZATA:

- CREARE E STILARE UN «button» E UN PANNELLO «ul» DI ELEMENTI «li»;
- IL CLICK SUL PULSANTE MOSTRA/NASCONDE IL PANNELLO;
- IL CLICK SU UN'OPZIONE NASCONDE IL PANNELLO E MOSTRA L'OPZIONE SCELTA (AGGIUNGERE UN ELEMENTO AL DOM);
- SE IL PANNELLO È APERTO, MUOVERSI CON LE FRECCE SU/GIU AGGIUNGE UN OUTLINE ALL'OPZIONE PRECEDENTE/SUCCESSIVA;
- APRIRE/CHIUDERE IL PANNELLO RESETTA L'OUTLINE.

TEMPO: 20 MINUTI

EVENTS - 28



Select a Subject

Mathematics

English

Physics

Select a Subject

Mathematics

English

Physics

FORMS - 1



IN HTML UN FORM È UN TAG CHE PERMETTE DI GESTIRE INPUT PROVENIENTI DALL'UTENTE; ESSO PUÒ CONTENERE ELEMENTI COME CAMPI DI TESTO, SELECT, TEXTAREA ETC.

PER CIASCUNO DI ESSI ESISTE OVVIAMENTE UN CORRISPONDENTE OGGETTO JAVASCRIPT CON PROPRIETÀ ED EVENTI SPECIFICI.

SI ACCEDE AI FORM NELLA PAGINA CON «`document.forms`», CHE È UNA COLLEZIONE ORDINATA DI TUTTI I FORM NELLA PAGINA; ALL'INTERNO DEL SINGOLO FORM SI ACCEDE AGLI ELEMENTI CON «`form.elements`».

FORMS - 2

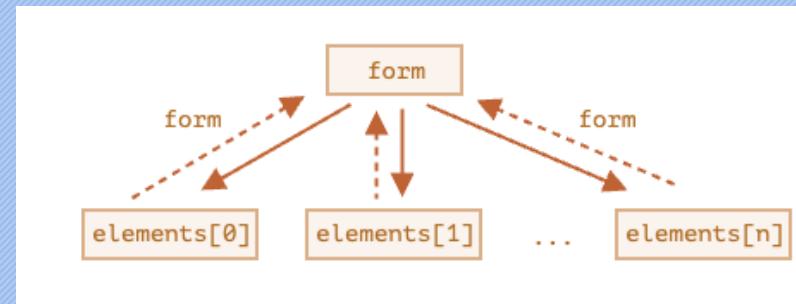
POSSONO ESSERCI PIÙ FORM CON LO STESSO «name», COME NEL CASO DEI RADIO BUTTON;
IN QUESTO CASO
«`forms.elements[name]`» È UNA COLLECTION.

I FORM POSSONO ANCHE AVERE UN SOTTOLIVELLO («`fieldset`»); IN QUESTO CASO SI DEVE ACCEDERE ALLA PROPERTY CORRISPONDENTE PRIMA DI ARRIVARE ALL'ELEMENT.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8      <form name="first">
9          <input name="zero" value="0">
10     </form>
11     <form name="second">
12         <input name="one" value="1">
13         <input name="two" value="2">
14     </form>
15
16     <script>
17         const forms = document.forms;
18         console.log(forms.length);
19         // 2
20
21         const firstObj = document.forms.first; // <form name="first"> element by name
22         const firstArr = document.forms[0]; // <form name="first"> element by array index
23         console.assert(firstArr === firstObj);
24
25         const elem = document.forms[1].elements.one; // <input name="one"> element
26         console.log(elem.value);
27         // 1
28     </script>
29
30 </body>
31 </html>
```

FORMS - 3

VICEVERSA, SI PUÒ ACCEDERE
ALL'OGGETTO «form»
ALL'INTERNO DEL SINGOLO
ELEMENT TRAMITE LA PROPERTY
CON LO STESSO NOME;
CIÒ RISULTA UTILE AD ES. PER LA
GESTIONE DEI MESSAGGI DI
ERRORE.



FORMS - 4



IL VALORE DEL SINGOLO ELEMENTO NEL FORM DIPENDE DAL TIPO DI ELEMENTO:

- INPUT DI TESTO: «`input.value`»;
- CHECKBOX / RADIO BUTTON: «`input.checked`»;
- TEXTAREA: «`input.value`»;
- SELECT: «`select.value`».

FORMS - 4.1



LA «select» È UN CAMPO UN PO' PARTICOLARE PER LA PRESENZA DELLE «option» AL SUO INTERNO;

ESSA INFATTI, OLTRE AL «value» ESPONE:

- UNA PROPRIETÀ «options» CHE RESTITUISCE LA LISTA CON TUTTE LE OPZIONI;
- UNA PROPRIETÀ «selectedIndex» CHE RESTITUISCE L'INDICE DELL'OPZIONE CORRENTEMENTE SELEZIONATA.

È POSSIBILE APPENDERE DINAMICAMENTE OPZIONI ALLA «select» TRAMITE L'OGGETTO CORRISPONDENTE «Option».

FORMS - 4.2



PER SETTARE IL VALORE DI UNA «select» SI PUÒ:

- SETTARE «selected='true'» SU UNA OPTION;
- SETTARE IL «selectedIndex» SULL'INDICE DELL'OPZIONE SCELTA;
- SETTARE DIRETTAMENTE IL «value» DELLA «select».

INOLTRE È POSSIBILE SETTARE PIÙ VALORI SE SULLA «select» È PRESENTE L'ATTRIBUTO «multiple».

FORMS - 5



GLI ELEMENTI IN UN FORM RICEVONO IL FOCUS QUANDO SI CLICCA SU DI ESSI O SI USA IL TASTO «Tab» DELLA TASTIERA;
UN ELEMENTO CON FOCUS È PRONTO A RICEVERE DATI.

VICEVERSA, SPOSTANDO IL FOCUS (BLUR) CLICCANDO ALTROVE O PREMENDO ANCORA «Tab» SI ASSUME CHE IL CAMPO SIA STATO VALORIZZATO E I DATI POSSONO ESSERE USATI.

GLI ELEMENTI HANNO DUE EVENTI CORRISPONDENTI A QUESTE DUE AZIONI, CHIAMATI APPUNTO «focus» E «blur».

FORMS - 6

I DUE EVENTI HANNO
«CAPTURING» MA «BUBBLING».
IN ALTERNATIVA AL «focus» SI
POSSONO USARE «focusin» E
«focusout».

I METODI CORRISPONDENTI
POSSENO ESSERE ANCHE
CHIAMATI VIA JAVASCRIPT PER
FORZARE IL COMPORTAMENTO
DELLA PAGINA.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6     <style>
7       .invalid { border-color: red; }
8       #error { color: red }
9     </style>
10    </head>
11    <body>
12
13      Your email please: <input type="email" id="input">
14      <div id="error"></div>
15
16    <script>
17      document.querySelector('#input').onblur = function() {
18        if (!input.value.includes('@')) { // not email
19          document.querySelector('#input').classList.add('invalid');
20          document.querySelector('#error').innerHTML = 'Please enter a correct email.'
21        }
22      };
23
24      document.querySelector('#input').onfocus = function() {
25        if (this.classList.contains('invalid')) {
26          // remove the "error" indication, because the user wants to re-enter something
27          this.classList.remove('invalid');
28          document.querySelector('#error').innerHTML = "";
29        }
30      };
31    </script>
32  </body>
33 </html>
```



SOLO ALCUNI ELEMENTI SPECIFICI AMMETTONO IL FOCUS PER DEFAULT, NELLO SPECIFICO QUELLI CON CUI L'UTENTE PUÒ INTERAGIRE (<button>, <input>, <select>, <a>, ...).

SI PUÒ FORZARE IL FOCUS SU UN ELEMENTO GENERICO USANDO L'ATTRIBUTO «tabindex»:

- «tabindex='-1'» INDICA CHE L'ELEMENTO RICEVE IL FOCUS SE IL METODO VIENE CHIAMATO SU DI ESSO VIA JS;
- «tabindex='0'» INSERISCE L'ELEMENTO TRA QUELLI FOCUSABILI E RAGGIUNGIBILI VIA «Tab» SENZA MODIFICARNE L'ORDINE;
- «tabindex='1', '2', ...» L'ELEMENTO È MESSO 'IN CODA' DOPO QUELLI A INDICE PIÙ BASSO; È RARAMENTO USATO.

ESERCIZIO

CREARE UN «div» DI DIMENSIONI FISSATE E CON DEL TESTO.

AL «focus» DEVE TRASFORMARSI IN UNA «textarea» MANTENENDO IL
CONTENTUTO.

AL «blur» O QUANDO SI PREME «Invio» DEVE RITORNARE UN «div».

TEMPO: 20 MINUTI.

FORMS - 9



OLTRE A «focus» E «blur» ESISTONO ALTRI EVENTI LEGATI AI CAMPI DI INPUT: I DUE PIÙ IMPORTANTI SONO L'«input» E IL «change».

L'EVENTO DI «input» VIENE LANCIATO QUANDO IL VALORE («value») DI UN «input», «select» O «textarea» CAMBIA A CAUSA DELL'AZIONE DELL'UTENTE (ES. CLICK SU «checkbox» O INSERIMENTO TESTO IN UNA «textarea»);
NON VIENE LANCIATO SE IL VALORE CAMBIA PROGRAMMATICAMENTE.

L'EVENTO DI «change» È SIMILE MA NON VIENE LANCIATO AD OGNI MODIFICA DEL «value»; NELLO SPECIFICO VIENE LANCIATO:

- PER «input» DI TIPO «checkbox» QUANDO L'ELEMENTO È SELEZIONATO/DESELEZIONATO;
- PER «input» DI TIPO «radio» QUANDO L'ELEMENTO È SELEZIONATO MA NON DESELEZIONATO;
- QUANDO SI SELEZIONA UN VALORE PER «select» E PER «input» DI TIPO «date»/«file»;
- PER «textarea» E ALTRI TIPI DI «input» QUANDO L'ELEMENTO PERDE IL FOCUS SE IL «value» È CAMBIATO.

ESERCIZIO

CALCOLARE UNA STIMA GREZZA DELLO STIPENDIO MENSILE NETTO A PARTIRE DALLA «RAL» E DAL NUMERO DI MENSILITÀ:

- CREARE UN FORM CON UN INPUT NUMERICO PER LA «RAL» E UNA SELECT CON TRE VALORI (12/13/14);
- IL NETTO SI CALCOLA DIVIDENDO I DUE VALORI E CONSIDERANDO IL 25% IN MENO (TASSE)
- VISUALIZZARE IL RISULTATO; ESSO DEVE CAMBIARE OGNI VOLTA CHE SI DIGITA UN VALORE PER LA RAL O SI CAMBIA IL NUMERO DI MESI NELLA SELECT.

TEMPO: 20 MINUTI.

L'ULTIMO EVENTO CHE VEDREMO È QUELLO DI «submit».

ESSO VIENE LANCIATO DALL'OGGETTO FORM IN TRE MODI:

- SE SI CLICCA UN «button» DI TIPO «submit»;
- SE SI PREME INVIO MENTRE SI EDITA UN CAMPO NEL FORM;
- SE SI CHIAMA ESPLICITAMENTE «form.requestSubmit()».

SE SUL FORM SONO DEFINITI GLI ATTRIBUTI «action» E «method» PARTE LA CHIAMATA VERSO IL SERVER COI I DATI DEL FORM.

AGGANCIANDO UN HANDLER FUNCTION SULL'EVENTO SI POSSONO VERIFICARE I DATI PRIMA DELL'INVIO, EVENTUALMENTE BLOCCANDOLO TRAMITE «preventDefault».

L'EVENTO «submit» E L'API «fetch» SONO ALLA BASE DELLA COMUNICAZIONE CLIENT-SERVER.