



UNIVERSITÀ
DEGLI STUDI
DI PALERMO



Introduzione a Javascript

Corso di Programmazione Web e Mobile
a.a. 2021/2022

Prof. Roberto Pirrone

Sommario

- Javascript: linguaggio di scripting a oggetti
 - Breve storia
 - Filosofia di programmazione
 - Come si inserisce uno script nella pagina web
- Variabili, tipi semplici e operatori
- Le stringhe, i numeri e la gestione delle funzioni matematiche
- Strutture di controllo del flusso del programma
- Funzioni

JavaScript – storia

- Originariamente sviluppato da Brendan Eich della Netscape Communications con il nome di *Mochan* e successivamente di *LiveScript*.
- In seguito è stato rinominato *JavaScript* ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Sun Microsystems.
- JavaScript è stato standardizzato per la prima volta il 1997 dalla ECMA con il nome *ECMAScript*. L'ultimo standard, di giugno 2016, è ECMA-262 Edition 7. È anche uno standard ISO.

ECMAScript

- 1998: ECMA-262, 1th edition
- 1999: ECMA-262, 2nd edition
- Giugno 2002: ECMA-262, 3rd edition, standard ISO/IEC 16262:2002
- Dicembre 2009: ECMA-262, 5th edition
- Giugno 2011: ECMA-262, 5.1th edition, standard ISO/IEC 16262:2011

- Una nuova edizione ogni anno dal 2015 al 2020 incluso

JavaScript – caratteristiche

- Linguaggio di scripting orientato agli *oggetti* e agli *eventi*
 - Utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi tramite funzioni di script invocate da *eventi* innescati a loro volta in vari modi dall'utente (mouse, tastiera, caricamento della pagina ecc...)
 - Descrive l'ambiente del client web tramite un apposito insieme di *oggetti* che possono essere manipolati dallo script
- Linguaggio interpretato
- Sintassi simile a Java, C, C++
- Debolmente tipizzato
 - I tipi delle variabili sono inferiti, in fase di assegnamento, dalla valutazione delle espressioni

Inserimento di uno script nella pagina web

```
<!DOCTYPE html>  
<html>
```

```
  <head>
```

```
    ...
```

```
    <script src="myscript.js"></script>
```

```
    <script>
```

```
      codice Javascript
```

```
    </script>
```

```
    <style></style>
```

```
  </head>
```

```
  <body>
```

```
    codice HTML
```

```
    <script src="myscript1.js"></script>
```

```
  </body>
```

```
</html>
```

Possibili alternative



Tipi di dati

- Tutti i tipi di dati in Javascript sono oggetti, derivati dalla classe base `Object`
 - `Number`
 - `String`
 - `Boolean`
 - `Date`
 - `Array`
 - `Function`
 - `Math`

Tipi di dati

- L'operatore `typeof` ritorna il tipo del dato di una espressione

<code>typeof "John"</code>	<code>// Returns string</code>
<code>typeof 3.14</code>	<code>// Returns number</code>
<code>typeof NaN</code>	<code>// Returns number</code>
<code>typeof false</code>	<code>// Returns boolean</code>
<code>typeof [1, 2, 3, 4]</code>	<code>// Returns object</code>
<code>typeof {name: 'John', age: 34}</code>	<code>// Returns object</code>
<code>typeof new Date()</code>	<code>// Returns object</code>
<code>typeof function () {}</code>	<code>// Returns function</code>
<code>typeof myCar</code>	<code>// Returns undefined (if myCar is not declared)</code>
<code>typeof null</code>	<code>// Returns object</code>

Tipi di dati

- Per questi tipi si definiscono dei letterali e non è necessario invocare il costruttore

- Letterali dei tipi semplici

// Number

23

-32.567e-10

Infinity // è un valore numerico

Nan // risultato di una forma

// indeterminata ad es. 0/0

// String

'pippo'

"pluto"

// Boolean

true

false

Tipi di dati

- Altri letterali delle stringhe

\'

\"

\\

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

Tipi di dati

- Letterali particolari
 - `null`, valore dell'oggetto nullo
 - `undefined`, identificatore dichiarato e non inizializzato

Variabili

- Una variabile si dichiara con
 - `var`, con ambito di visibilità globale
 - `let`, con ambito di visibilità locale rispetto al blocco di definizione

```
var foo, bar = 12;
```

```
console.log('bar: ' + bar);  
console.log('foo: ' + foo);
```

```
if (!foo) {  
    let baz = bar;  
    console.log('baz: ' + baz);  
    //let baz = 2;  
}
```

```
var bar = "pippo";  
console.log('bar: ' + bar);  
console.log('baz: ' + baz);
```

Costanti

- Una variabile si dichiara con `const`

```
const myNumber = 123456;
```

Operatori - aritmetici

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (<u>ES2016</u>)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Operatori - assegnamento

Operator	Example	Same As
=	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

Operatori - confronto

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Operatori - logici

Operator	Description
&&	logical and
	logical or
!	logical not

Operatori – bit a bit

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
	OR	5 1	0101 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

Operatori – identificazione del tipo

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Per una lista completa con precedenza e associatività, si consulti il portale [Javascript MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators)

Conversioni automatiche di tipo

- Javascript è caratterizzato dal fatto che determina i tipi al momento del calcolo delle espressioni.
- La *type coercion* è l'insieme dei meccanismi automatici per effettuare la conversione automatica
 - I risultati non sono sempre immediatamente predicibili
 - Quanto fanno "5" + 1 e "5" * 1 ?

Conversioni automatiche di tipo

- Per quanto riguarda gli operatori booleani in cascata per la valutazione di più espressioni essi usano una *valutazione cortocircuito*
- Gli operandi sono valutati da sinistra a destra *andando avanti solo fino a quando non vi è certezza del risultato*
- La parte restante dell'espressione non è valutata

Number

`Number.toLocaleString`

`Number.valueOf`

`Number.apply`

`Number.call`

`Number.constructor`

`Number.toString`

`Number.EPSILON`

`Number.MIN_SAFE_INTEGER`

`Number.NEGATIVE_INFINITY`

`Number.NaN`

`Number.isInteger`

`Number.isSafeInteger`

`Number.length`

`Number.parseInt`

`Number.arguments`

`Number.caller`

`Number.MAX_SAFE_INTEGER`

`Number.MIN_VALUE`

`Number.POSITIVE_INFINITY`

`Number.isNaN`

`Number.name`

`Number.prototype`

`Number.bind`

`Number.MAX_VALUE`

`Number.isFinite`

`Number.parseFloat`

Number

```
x=new Number(8)  
[Number: 8]  
> x.
```

```
x.constructor  
x.toFixed  
x.toPrecision  
x.valueOf
```

```
x.toExponential  
x.toLocaleString  
x.toString
```

String

```
s=new String('pippo')  
[String: 'pippo']  
> s.
```

s.toLocaleString

s.anchor

s.charAt

s.constructor

s.fontsize

s.lastIndexOf

s.matchAll

s.repeat

s.small

s.sub

s.toLocaleLowerCase

s.toUpperCase

s.trimRight

s.big

s.charCodeAt

s.endsWith

s.includes

s.link

s.normalize

s.replace

s.split

s.substr

s.toLocaleUpperCase

s.trim

s.trimStart

s.blink

s.codePointAt

s.fixed

s.indexOf

s.localeCompare

s.padEnd

s.search

s.startsWith

s.substring

s.toLowerCase

s.trimEnd

s.valueOf

s.bold

s.concat

s.fontcolor

s.italics

s.match

s.padStart

s.slice

s.strike

s.sup

s.toString

s.trimLeft

Math

Math.

Math.E

Math.LOG2E

Math.abs

Math.asinh

Math.cbrt

Math.cosh

Math.fround

Math.log10

Math.min

Math.sign

Math.tan

Math.LN10

Math.PI

Math.acos

Math.atan

Math.ceil

Math.exp

Math.hypot

Math.log1p

Math.pow

Math.sin

Math.tanh

Math.LN2

Math.SQRT1_2

Math.acosh

Math.atan2

Math.clz32

Math.expm1

Math.imul

Math.log2

Math.random

Math.sinh

Math.trunc

Math.LOG10E

Math.SQRT2

Math.asin

Math.atanh

Math.cos

Math.floor

Math.log

Math.max

Math.round

Math.sqrt

Strutture di controllo del flusso del programma

- Scelta tra due alternative

```
if ( condizione ) {  
    codice da eseguire se la condizione è vera  
} else {  
    codice da eseguire se la condizione è falsa  
}
```

Strutture di controllo del flusso del programma

- Scelta tra due alternative (continua)

```
if ( condizione ) {  
    codice da eseguire se la condizione è vera  
}
```

Strutture di controllo del flusso del programma

- Scelta tra più alternative

```
if ( 1^ condizione ) {  
    codice da eseguire se la 1^ condizione è vera  
} else if ( 2^ condizione ){  
    codice da eseguire se la 2^ condizione è vera mentre la prima  
    è falsa  
} else if ...  
...  
} else {  
    codice da eseguire se tutte le condizioni precedenti  
    sono false  
}
```

Strutture di controllo del flusso del programma

- Scelta tra più alternative

```
switch( espressione da valutare una volta e che ritorna un valore ) {  
    case valore1:  
        codice da eseguire se l'espressione assume valore pari a valore1  
        break;  
    case valore2:  
        codice da eseguire se l'espressione assume valore pari a valore2  
        break;  
    ...  
    default:  
        codice da eseguire se l'espressione non assume nessuno dei valori previsti  
}
```

Strutture di controllo del flusso del programma

- Iterazioni

```
for ( inizializzazione; termine; aggiornamento) {  
    codice da eseguire  
}
```

Strutture di controllo del flusso del programma

- Iterazioni

```
while( condizione di iterazione ) {  
    codice da eseguire se la condizione è vera  
}
```

Strutture di controllo del flusso del programma

- Iterazioni

```
do {
```

```
    codice da eseguire
```

```
} while ( condizione di iterazione )
```


Strutture di controllo del flusso del programma

- `break` forza l'uscita dall'iterazione
- `continue` forza l'iterazione successiva

Strutture di controllo del flusso del programma

- Oggetti iterabili
 - Sono quegli oggetti che si prestano strutturalmente ad essere scanditi elemento per elemento:
 - Array, String, NodeList (classe del DOM), oggetti definiti dall'utente ...

for (*chiave* in *iterabile*) – itera sugli indici/chiavi/proprietà

Strutture di controllo del flusso del programma

- Oggetti iterabili
 - Sono quegli oggetti che si prestano strutturalmente ad essere scanditi elemento per elemento:
 - Array, String, NodeList (classe del DOM), oggetti definiti dall'utente ...

for (*elemento* on *iterabile*)

– itera sui valori di un oggetto iterabile con struttura assimilabile agli

Array (*non* gli oggetti definiti dall'utente) generando così un *iteratore*

Strutture di controllo del flusso del programma

- *Iteratore*
 - Struttura dati che consente di accedere agli elementi di una sequenza attraverso un'unica operazione di accesso all'elemento elemento, senza possibilità di reinizializzare la sequenza

Funzioni

- Dichiarazione

```
function nomeFunzione(parametro, parametro, ...) {  
    codice da eseguire  
    return risultato  
}
```

- Se non c'è return la funzione non ritorna nulla, cioè il risultato è void

Funzioni

- Una funzione è un oggetto di tipo `Function`, quindi è una variabile che può essere dichiarata anche

```
var nomeFunzione = function (parametro, ...) {  
    codice da eseguire  
    return risultato  
}
```

Funzioni

- Dichiarazione di *funzione arrow*

```
var nomeFunzione = (parametro, ...) => {  
    codice da eseguire  
    return risultato  
}
```

- Buona notazione per piccole funzioni

Funzioni

- Il passaggio dei parametri è per valore, quindi con copia locale dell'oggetto passato come parametro

```
function f(x) {  
    x = x.replace(/p/g, '*');  
    console.log(x);  
}
```

```
> var y = 'Pippo'  
'Pippo'
```

```
> f(y)  
'Pi**o'
```

```
> y  
'Pippo'
```


Funzioni

- Closure:
 - La funzione fa da wrapper ad un'altra funzione, restituita come risultato, che conserva al suo interno eventuali variabili locali che si vogliano conservare

```
function wrapValue(n) {  
    let local = n;  
    return () => local;  
}
```

```
> let wrap1=wrapValue(3)  
undefined  
> let wrap2=wrapValue(5)  
undefined  
> wrap1()  
3  
> wrap2()  
5
```

Oggetti

- Dichiarazione

```
var nomeOggetto = {  
  proprietà: valore,  
  proprietà: valore,  
  metodo: function(parametro, parametro, ...) { codice },  
  ...  
}
```

Oggetti

- Gli oggetti definiti dall'utente sono di classe `Object`
- Stringhe, array, date, espressioni regolari, numeri, booleani sono oggetti predefiniti in Javascript
- Accesso ad un elemento di un oggetto

nomeOggetto["proprietà"]

nomeOggetto.proprietà

nomeOggetto.metodo(lista parametri)