ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Introduzione a Javascript I parte

Fabio Vitali

Corsi di laurea in Informatica e
Informatica per il Management

Alma Mater – Università di Bologna

Oggi parleremo di…

Javascript

- ***Sintassi base (parte I)***
- Modello oggetti del browser (parte II)

AJAX (parte III):

- Architettura di riferimento
- XMLHttpRequest

ECMAScript

- Linguaggio di script client-side.
- Nato nel 1995 con il nome di JavaScript introdotto da Sun e Netscape
- Standardizzato da ECMA International nel 1997. Per 18 anni sono state fatte piccole estensioni e modifiche.
- Nel giugno 2015 è stata approvata la versione 6 (nota anche come EcmaScript 2015). Con la forte pressione di WHATWG, sono state introdotte molte differenze, molti nuovi costrutti.
- Nel giugno 2022 è stata approvata la versione 13, EcmaScript 2022. Alcuni costrutti nuovi, tra cui top-level await, campi e metodi privati, campi e metodi statici.
- ES.Next è un nome dinamico che fa riferimento a feature già implementate mentre vengono discusse, e prima o poi rilasciate in una versione ufficiale di EcmaScript.
- Fa parte dell'approccio *living standard* del WHATWG

Come eseguire uno script Javascript

- Client-side: eventi
  - Ogni elemento del documento ha alcuni eventi associati (click, mouseover, doppio click, tasto di tastiera, ecc), e degli attributi ***on+evento*** associati.
  - inserendo istruzioni JavaScript (o chiamata a funzione) nel valore dell'attributo si crea una *chiamata callback*.
  - Ci sono due eventi particolari da aggiungere: ***load*** (della window) e ***ready*** (del documento). Ne riparleremo.
- Server-side: routing
  - I servizi server-side sono associati a URI. Il modo più antico e semplice è creare servizi diversi e inserirli in file separati, ciascuno con un URI proprio. Aprendo una connessione all'URI, viene invocato lo script ed eseguito il servizio.
  - NodeJs (Express.js, in realtà) fornisce un meccanismo per associare una funzione javascript (callback) ad ogni tipo di URI. Aprendo una connessione all'URI, lo script centrale esegue la funzione corrispondente.

Come eseguire uno script sul browser

- *In maniera sincrona*, appena lo script viene letto, in un tag <script> o in un file.
  - Adatto per inizializzare oggetti e variabili da usare più tardi.
- *In maniera asincrona*, associando il codice ad un evento sul documento (e.g., il click su un bottone): *event-oriented processing*
  - Il tipo più comune di script Javascript
- *In maniera asincrona*, associando il codice al completamento di un'operazione di rete.
  - Le operazioni Ajax vengono gestite tramite *callback* , funzioni eseguite appena la richiesta HTTP asincrona è completata e i dati sono stati ricevuti.
- *In maniera asincrona*, associando il codice ad un *timeout*, i.e., un periodo di attesa dopo il quale lo script viene eseguito automaticamente.
- Durante l'esecuzione dello script, il browser è bloccato e non reagisce agli input dell'utente. Per questo gli scipt debbono essere brevi e veloci in modo da lasciare all'utente la sensazione di interattività e controllo.

Come mandare in output gli script

- HTML visualizza l'output degli script in 4 modi diversi:
  - scrivendo direttamente nella finestra del browser:
    ```
    document.write(string) ;
    ```
  - scrivendo sulla console:
    ```
    console.log(string) ;
    ```
  - scrivendo in una finestra di alert:
    ```
    alert(string) ;
    ```
  - modificando il DOM del documento visualizzato:
    ```
    document.getElementById(id).innerHTML = string ;
    ```

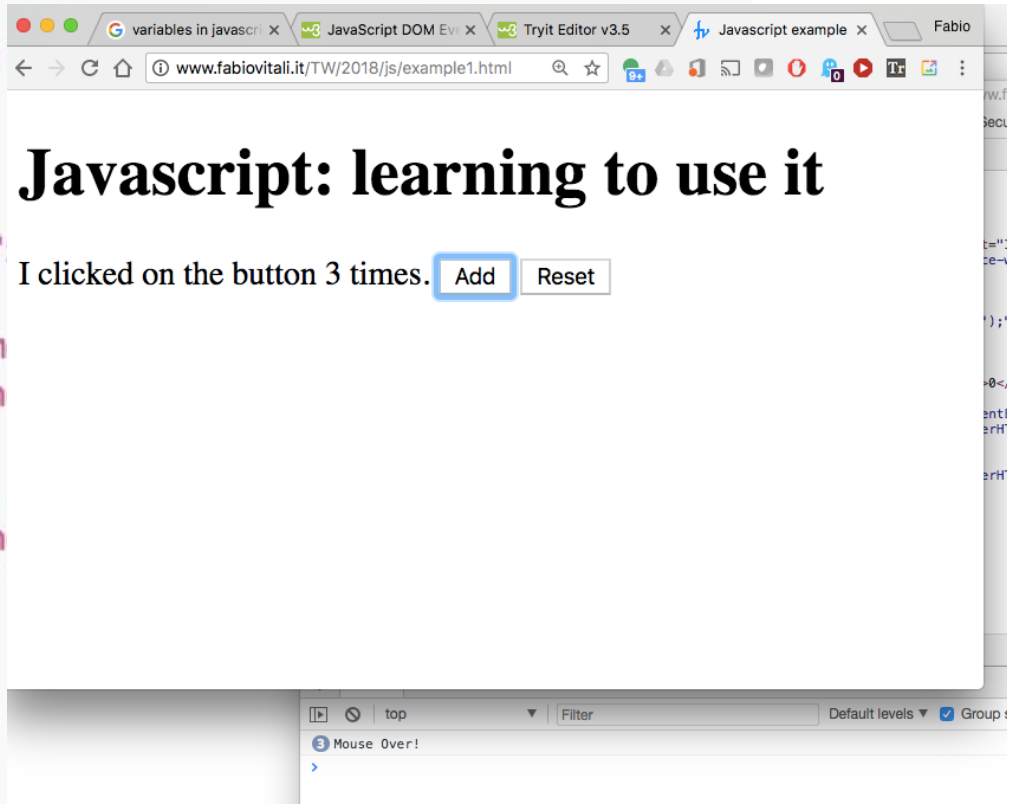
- I prossimi esercizi si trovano in
  `http://www.fabiovitali.it/TW/2022/js/`

Come attivare gli script

- HTML prevede l'uso di script in tre modi diversi
  - posizionato dentro all'attributo di un evento
  - posizionato nel tag <script>
  - indicato in un file esterno puntato dal tag <script>

Posizionato dentro all'attributo di un evento

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Javascript example</title>
    </head>
    <body onLoad="alert('hello');">
        <h1 onMouseOver="console.log('Mouse Over!
            Javascript: learning to use it
        </h1>
        <p id="p1">
            I clicked on the button <span id="s1"
            <button onclick="
                var n = parseInt(document.getElem
                document.getElementById('s1').inn
            ">Add</button>
            <button onclick="
                document.getElementById('s1').inn
            ">Reset</button>
        </p>
    </body>
</html>
```

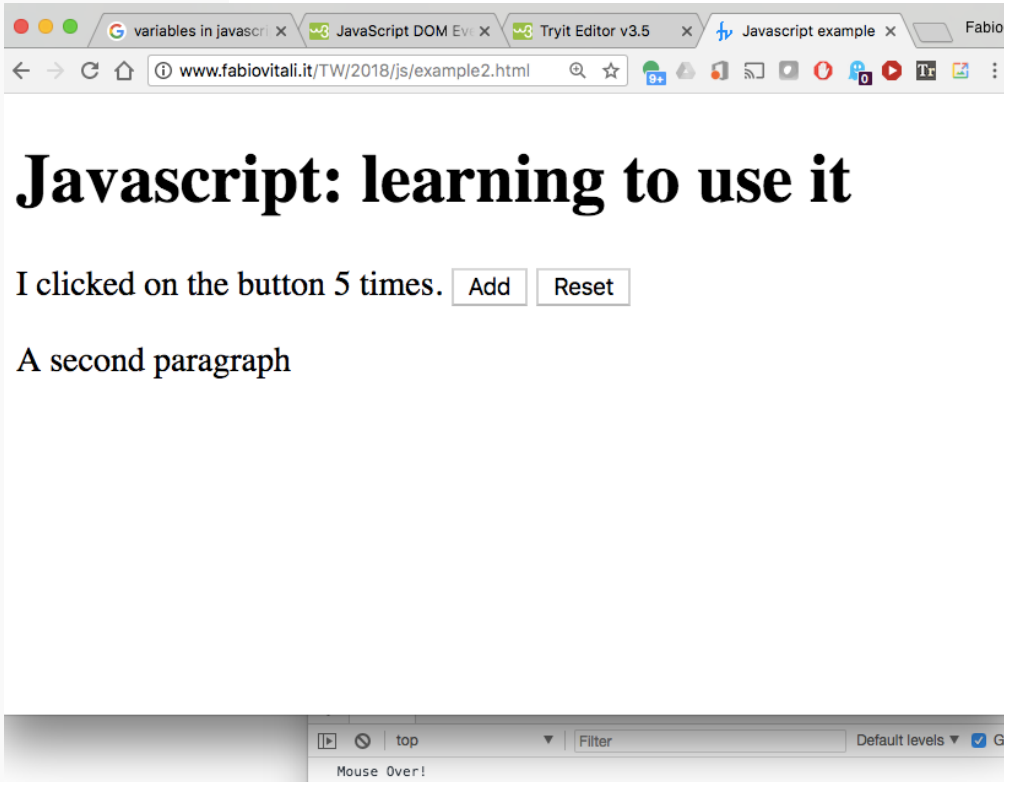


Posizionato nel tag <script>

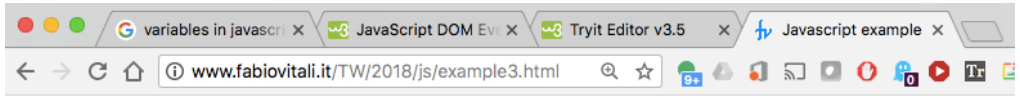
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Javascript example</title>
        <script type="text/javascript">
            function hello() {
                alert('hello');
            }

            function add() {
                var n = parseInt(document.getElementById('s1').innerHTML);
                document.getElementById('s1').innerHTML = n+1;
            }

            function reset() {
                document.getElementById('s1').innerHTML = 0;
            }
        </script>
    </head>
    <body onLoad="hello()">
        <h1 onMouseOver="console.log('Mouse Over!');">
            Javascript: learning to use it
        </h1>
        <p id="p1">
            I clicked on the button <span id="s1">0</span> times.
            <button onclick="add()">Add</button>
            <button onclick="reset()">Reset</button>
        </p>
        <script type="text/javascript">
            document.write("<p>A second paragraph</p>") ;
        </script>
    </body>
</html>
```



Indicato dal tag <script>



A second paragraph

Javascript: learning to use it

I clicked on the button 2 times. Add Reset

```
function hello() {
    alert('hello');
}

function add() {
    var n = parseInt(document.getElementById('s1').innerHTML);
    document.getElementById('s1').innerHTML = n+1;
}

function reset() {
    document.getElementById('s1').innerHTML = 0;
}

document.write("<p>A second paragraph</p>") ;
```

```
<
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Javascript example</title>
        <script type="text/javascript" src="script1.js"> </script>
    </head>
    <body onLoad="hello()">
        <h1 onMouseOver="console.log('Mouse Over!');">
            Javascript: learning to use it
        </h1>
        <p id="p1">
            I clicked on the button <span id="s1">0</span> times.
            <button onclick="add()">Add</button>
            <button onclick="reset()">Reset</button>
        </p>
    </body>
</html>
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Javascript base

(In che modo Javascript è ***simile*** agli altri linguaggi)

JS: Tipi di dato

- Javascript è minimale e flessibile per quel che riguarda i tipi di dati.
- Ci sono tre importanti tipi di dati atomici built-in:
  - booleani
  - numeri (sia interi, sia floating point)
  - stringhe
- Inoltre vanno considerati come tipi di dati anche
  - null
  - undefined
- C'è poi un unico tipo di dato strutturato, *object*, di cui fanno parte anche gli *array*.

JS: Variabili

- I dati in Javascript sono tipati, ma le variabili no.

```
var pippo ;
pippo = "ciao" ;
pippo = 15;
pippo = [1, 2, 3] ;
```

- Due modi per definire variabili
  - `var pippo='ciao' ;` definisce una variabile nello scope della funzione o del file in cui si trova.
  - `let pippo='ciao';` definisce una variabile nello scope del blocco parentetico o della riga in cui si trova.
  - `const pippo='ciao';` definisce una variabile non ulteriormente modificabile.

JS: Operatori

Numeri

Operatore	Descrizione	Esempio	Commento
+	Somma	`let a = 5 + 7`	a vale 12
-	Sottrazione	`let b = 17 - 2`	b vale 15
*	Moltiplicazione	`let c = 5 * 4`	c vale 20
/	Divisione	`let d = 28 / 4`	d vale 7
%	Modulo	`let e = 15 % 6`	e vale 3 (15 / 6 = 2 resto 3)
**	Esponente	`let f = 3**2`	f vale 9 (cioè 3^2)
++	Incremento	`e++`	e vale 4 (3 + 1)
--	Decremento	`f--`	f vale 8 (9 - 1)

Stringhe

+	composizione	`g = "Hel"+"lo"`	g vale "hello"
+	composizione + casting	`h = "5" + 7`	h vale "57"

JS: Operatori

Confronto e booleani

Operatore	Descrizione	Esempio	Commento
==	Uguaglianza	`let i = b==c` `let j = 5=='5'`	i è falso (b vale 15 e c vale 20) j è vero (con casting di 5 in '5')
<	Minore	`let k = b<c`	k è vero
>	Maggiore	`let l = b>c`	l è falso
<=	Minore o uguale	`let m = b<=c`	m è vero
>=	Maggiore o uguale	`let n = b>=c`	n è falso
!=	Disuguaglianza	`let o = b!=c` `let p = 5!='5'`	o è vero p è falso (con casting di 5 in '5')
===	Uguaglianza senza casting	`let q = 5==='5'`	q è falso (non avviene casting di 5 in '5')
!==	Disuguaglianza senza casting	`let r = 5!=='5'`	r è vero (non avviene casting di 5 in '5')
&&	AND	`let s = i&&j`	s è falso
||	OR	`let t = i||j`	t è vero
!	NOT	`let u = !j`	u è falso

JS: Strutture di controllo condizionali (1)

- Blocco if

```
if (a==5)
  istruzione_singola ;
```

```
if (a==5) {
  istruzione_1;
  istruzione_2;
  ...
}
```

```
if (a==5) {
  istruzione_1;
  istruzione_2;
  ...
} else {
  istruzione_3;
  istruzione_4;
  ...
}
```

JS: Strutture di controllo condizionali (2)

- Operatore ternario

```
let x = (b==5 ? 'pippo' : 'paperino') ;
```

- Blocco switch

```
switch (a) {
case 'a':
  istruzione_1; istruzione_2 ;
  ... ;
  break;
case 'b':
  istruzione_3; istruzione_4 ;
  ... ;
  break;
...
default:
  istruzione_n; istruzione_npiu1 ;
  ...;
  break;
}
```

JS: strutture di controllo
cicli (1)

- Blocco for

```
for (let i=0; i<k; i++) {
    istruzione_1;
    istruzione_2;
    alert(i);
    ....
}
```

i è l'indice di controllo del loop. E' un intero

- Blocco for ... in

```
for (j in obj) {
    istruzione_1;
    istruzione_2;
    alert(obj[j]);
  ....
}
```

j è l'indice di controllo del loop. E' una stringa

JS: strutture di controllo
cicli (2)

- Blocco while

```
while (k < 5) {
    istruzione_1;
    istruzione_2;
    ...
}
```

- Blocco do ... while

```
do {
    istruzione_1;
    istruzione_2;
    ...
} while (k < 5)
```

JS: strutture di controllo
eccezioni

- Blocco try ... catch

```
let x=prompt("Enter a number between 0 and 9:","");

try {
    let el = document.getElementById("menu"+x)
    let address = el.attributes["href"].value
    return address ;
} catch(er) {
    return "input Value out of bounds" ;
}
```

JS: funzioni

- Le funzioni in Javascript sono blocchi di istruzioni dotati di un nome e facoltativamente di parametri.
- Possono ma non sono obbligate a restituire un valore di ritorno. Le funzioni non sono tipate, i valori di ritorno sì (come al solito).

```
function double(n) {
  let m = n+n;
  return m ;
}
let a = double(4) ;
let b = double('test') ;
```

a vale 8
b vale 'testtest'

- Se manca un parametro, non restituisce errore ma assume che il parametro sia undefined.

```
function double(n) {
  if (typeof n!= undefined) {
    let m = n+n;
  } else {
    let m = 0 ;
  }
  return m ;
}
let c = double() ;
```

```
function double(n) {
  return n? n+n : 0;
}
```

c vale 0

C'è molto di più da dire sulle funzioni. Ne riparliamo presto.

JS: Tipi di dati strutturati (1)

- Javascript ha un unico tipo di dato strutturato, chiamato *object*. Anche gli array sono un tipo speciale di object.
- Gli object sono liste non ordinate di proprietà, coppie nome-valore.

```
let persona = {
  nome:    'Giuseppe',
  cognome: 'Rossi',
  altezza: 180,
  nascita: new Date(1995,3,12)
}
```

JS: Tipi di dati strutturati (2)

- Il valore di una proprietà può essere esso stesso un object.

```
let persona = {
  nome:    'Giuseppe',
  cognome: 'Rossi',
  altezza: 180,
  nascita: new Date(1995,3,12),
  indirizzo: {
     via: {
        strada: 'Via Indipendenza',
        numero: '15'
     },
     citta: 'Bologna',
     nazione: 'Italia'
  }
}
```

- E' un (frequente) errore la virgola dopo l'ultimo elemento di un blocco. Alcuni interpreti la ignorano, altri no.

JS: Tipi di dati strutturati (3)

Ci sono due sintassi per accedere alle proprietà di un object:

- *Dot syntax* (ispirazione dai linguaggi Object Oriented)

```
alert(persona.nome + ' ' + persona.cognome)
```

- *Square bracket syntax* (ispirazione dagli array associativi)

```
alert(persona['nome'] + ' ' + persona['cognome'])
```

il nome della proprietà in questo caso è una normalissima stringa:

```
let n1 = 'nome' ;
alert(persona[n1]);
```

posso anche fare elaborazioni sulle stringhe:

```
let n2 = 'cog'+ n1;
alert(persona[n1]+' '+persona[n2]);
```

JS: Tipi di dati strutturati (4)

Uso entrambe le sintassi per leggere e per scrivere le proprietà dell'object:

```
persona.cognome = 'Verdi' ;
persona['nome'] = 'Antonio' ;

let n1 = 'nome' ;
persona[n1] = 'Andrea' ;
```

Moltiplico i punti o aggiungo square bracket per proprietà in oggetti annidati:

```
persona.indirizzo.via.numero = '36' ;
persona['indirizzo']['via']['numero'] = '15/a' ;
```

JS: Array (1)

Un array è un object in cui le chiavi sono numeri interi assegnati automaticamente. Per distinguerlo da un oggetto normale usa la parentesi quadra invece che la graffa.

```
let nomi = ['Andrea', 'Beatrice', 'Carlo'] ;
```

La dot syntax non può essere usata, ma solo quella bracket. Il primo numero è lo zero.

```
alert(nomi[0] + ' ' + nomi[1]) ;
```

Posso normalmente leggere e scrivere elementi dell'array:

```
nomi[0] = 'Adriano';
```

JS: Array (2)

let nomi = ['Andrea, 'Beatrice', 'Carlo']

Un array è un object con alcuni metodi e proprietà molto utili:

- **length**: lunghezza dell'array

```
let n = nomi.length;
```

n == 3

- **indexOf(item):** la posizione di *item* nell'array

```
let k = nomi.indexOf('Beatrice')
```

k == 1

- **pop()**: toglie un valore in fondo all'array e lo restituisce

```
let c = nomi.pop()
```

c == 'Carlo', *nomi* == ['Andrea', 'Beatrice']

- **push(item)**: aggiunge *item* in fondo all'array

```
nomi.push('Davide')
```

nomi == [Andrea', 'Beatrice', 'Davide']

- **shift()**: toglie un valore in cima all'array e lo restituisce

```
let a = nomi.shift()
```

a == 'Andrea', *nomi* == ['Beatrice', 'Davide']

- **unshift(item)**: aggiunge *item* in cima all'array e restituisce la nuova lunghezza

```
let d = nomi.unshift('Antonio')
```

nomi == ['Antonio', 'Beatrice', 'Davide']

JS: Array (3)

Altri metodi utili:

nomi == ['Antonio', 'Beatrice', 'Davide']

- **slice**(start,end): restituisce un array da start a end (escluso):

```
let b = nomi.slice(1,2);
```

b == ['Beatrice']

- **splice(**pos,rimuovi,inserisci**):** inserisce e rimuove elementi

```
let pers = ["Andrea", "Barbara", "Carlo", "Elena"];
pers.splice(2, 1, "Claudio");
```

- *Rimosso 1 elemento dalla posizione 2 ('Carlo')*
- *Aggiunto un nuovo item*

persone == ["Andrea", "Barbara", **"Claudio"**, "Elena"]

- **join(**sep**):** crea una stringa usando *sep* come separatore.

```
let p = pers.join(", ");
```

p == "Andrea, Barbara, Claudio, Elena"

JS: Array (4)

Oggetti e array possono contenersi liberamente. Attenzione ad usare parentesi quadre per gli array e graffe per gli oggetti.

```
let persona = {
  nome:    ['Giuseppe', 'Andrea', 'Federico'],
  cognome: 'Rossi',
  altezza: 180,
  nascita: new Date(1995,3,12),
  indirizzo: {
     via: {
        strada: 'Via Indipendenza',
        numero: '15'
     },
     citta: 'Bologna',
     nazione: 'Italia'
  },
  telefono: [
     { tipo: 'casa', numero: '051 123456'},
     { tipo: 'cell', numero: '335 987654'}
  ]
}
```

JS: Oggetti Predefiniti

- Javascript predefinisce alcuni oggetti utili per raccogliere insieme i metodi più appropriati per certi tipi di dati.
- Oggetti multipli
  - Object
  - Array
  - String
  - Date
  - Number
  - RegExp
  - ...
- Oggetti singoletto
  - Math
  - JSON
  - ...

JS: Stringhe

L'oggetto String contiene metodi disponibili per tutti i valori di tipo stringa:

```
let str = 'Precipitevolissimevolmente';
```

- **length**: lunghezza della stringa

```
let s = str.length;
```

s == 26

- **indexOf(**sub**):** la posizione di *sub* nella stringa

```
let t = str.indexOf('ss')
```

t == 13

- **substring(**start, end**)**: restituisce la sottostringa da start ad end

```
let x = str.substring(3,8)
```

x == 'cipit'

- **substr(**start, length**)**: restituisce la sottostringa da start per length caratteri

```
let y = str.substr(3,8)
```

y == "cipitevo"

- **split(**sep**)**: separa una stringa in array di utilizzando sep come separatore.

```
let w = "130.136.1.110" ;
let z = w.split(".") ;
```

z == ['130', '136', '1', '110']

JS: JSON (1)

JSON (*JavaScript Object Notation*) è un formato dati derivato dalla notazione usata da JS per gli oggetti.

```
{
  "nome":    ["Giuseppe", "Andrea", "Federico"],
  "cognome": "Rossi",
  "altezza": 180,
  "nascita": "1995-04-11T22:00:00.000Z",
  "indirizzo": {
     "via": {
        "strada": "Via Indipendenza",
        "numero": "15"
     },
     "citta": "Bologna",
     "nazione": "Italia"
  },
  "telefono": [
     { "tipo": "casa", "numero": "051 123456"},
     { "tipo": "cell", "numero": "335 987654"}
  ]
}
```

JS: JSON (2)

- Rispetto alla notazione usata nei programmi, bisogna ricordare solo:
  - Solo valori string, number, boolean, array o object
  - Anche i nomi delle proprietà sono tra virgolette
  - Si usano solo le virgolette doppie e non le semplici
  - Non si possono inserire commenti di nessun tipo
- Tutti i linguaggi di programmazione più importanti oggi accettano e scrivono dati in JSON

JS: JSON (3)

JSON è un singoletto in JavaScript che supporta due soli metodi.

```
let j = {
    nome: 'Fabio',
    voto:10
};

let s = JSON.stringify(j);
let t = JSON.stringify(j, null, 2);


let k = '{
    "cognome": "Rossi",
    "amici": ["Andrea", "Lucia"]
}' ;

let u = JSON.parse(k) ;
```

s vale la stringa: `'{ "nome": "Fabio", "voto":10}'`

t vale la stringa
```
'{
  "nome": "Fabio",
  "voto": 10
}'
```

u vale l'oggetto
```
{
  cognome: "Rossi",
  amici: ["Andrea", "Lucia"]
}
```

JS: Date

Una data è un oggetto che esprime un giorno e un orario rappresentandolo come il numero di millisecondi trascorsi dalla mezzanotte del 1 gennaio 1970 e la data in questione. Poiché in realtà è un numero, questo permette di fare operazioni aritmetiche e confronti numerici.

- **Costruttore:**

```
let d = new Date();
let Capodanno = new Date(2022,0,1);
```

d contiene la data e l'ora di adesso

I mesi sono contati da 0

- **getDay():** Il giorno della settimana della data (0-Domenica fino a 6-Sabato)

```
let w = Capodanno.getDay()
```

w vale 6 (Sabato)

- **toDateString()**: converte il numero in una stringa leggibile

```
let a = Capodanno.toLocaleDateString()
let b = Capodanno.toDateString()
```

"01/01/2022"

"Sat Jan 01 2022"

- ***Operazioni***

```
let msInADay = 1000*60*60*24;
let msSinceCapodanno = d - Capodanno ;
let daysSinceCapodanno = Math.round(msSinceCapodanno/msInADay)
```

daysSinceCapodanno == 79

- ***Confronti***

```
let In2022 = d > Capodanno
```

In2022 è vero

Altri oggetti

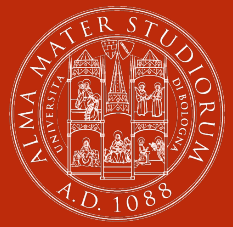
- Math
  - Singoletto che raccoglie funzioni e costanti matematiche utili
  - `Math.PI, Math.abs(), Math.sin() Math.cos(), Math.round(), Math.sqrt(),Math.random`, ecc.
- RegExp
  - Classe delle espressioni regolari, da usare per fare match su stringhe. Le RegExp usano '/' come delimitatore.

```
let str = 'Precipitevolissimevolmente' ;
let re = /pi(.)/ ;
let x = str.match(re) ;
let y = re.exec(str) ;
```

sia x, sia y sono array
con due match: ['pit' e 't']

Esercizi

- Semplici esercizi su http://www.fabiovitali.it/TW/2023/js/



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fabio Vitali

Dipartimento di Informatica – Scienze e Ingegneria
Alma mater – Università di Bologna

Fabio.vitali@unibo.it

www.unibo.it