# Gestione delle date in Java Script

Rev. 1.2 del 21/05/2022

Siccome il web è utilizzato a livello planetario, l'oggetto javascript **ISODate** o semplicemente **Date** (standard ISO 8601 del 1988) memorizza le date **SEMPRE** con l'ora riferita al meridiano 0 di Greenwich (il cosiddetto **Greenwich Mean Time**). In Italia, che adotta l'ora standard dell'Europa centrale, l'ora solare invernale è +0**1**:00 rispetto al GMT, quella legale estiva +0**2**:00 rispetto al GMT.

- La data viene salvata all'interno dell'oggetto Date come numero intero a **64 bit** che rappresenta il cosiddetto timestamp unix, cioè il numero di **millisecondi** trascorsi dalla data zero dei sistemi unix (1/1/1970).
- Data e ore memorizzate in un oggetto **Date** sono *SEMPRE* riferite al GTM
- I numeri negativi indicano le date antecedenti al 1/1/1970.
- Su 64 bit sono rappresentabili 290 milioni di anni sia in positivo che in negativo.

## Creazione di un oggetto Date

Per creare una nuovo **oggetto Date** si utilizza semplicemente il costruttore dell'oggetto:

```
var dataCorrente = new Date();
```

che restituisce la data corrente come oggetto **ISODate**.
Il metodo `Date()` **senza il new** restituisce la data corrente come semplice stringa !

E' anche possibile creare un oggetto **ISODate** a partire da una generica data scritta come stringa in formato **UTC**. Gli **UTC Format** validi sono i seguenti:

```
YYYY
YYYY-MM
YYYY-MM-DD
MM-DD-YYYY
YYYY-MM-DDThh:mm±HH:MM     (L'ora viene interpretata come ora locale)
YYYY-MM-DDThh:mm:ss±HH:MM
YYYY-MM-DDThh:mm:ss.mmm±HH:MM
mmm di un timestamp unix   (Intesi come ora GMT. Es 1651835027598 => 2022-05-05T13:00:00.000Z)
```

## Esempi di creazione di date e gestione del timezone

```
var dataDiNascita = new Date("2020-10-13");
var dataCompleta = new Date("2020-10-13T00:00:00");
```

In fase di creazione dell'oggetto Date:

- Se il time viene omesso, viene automaticamente assunta come ora **l'ora zero di Greenwich**
- Se il time viene indicato esplicitamente, viene interpretato come ora locale. In questo caso la funzione new Date() richiede al sistema operativo il timezone della macchina e, sulla base del timezone, provvede a ricalcolare data e ora come "**timestamp assoluto riferito a Greenwich"** e a memorizzarle come Greenwich time. In qualunque momento, le funzioni di visualizzazioni potranno leggere il timezone dal Sistema Operativo e ricalcolare l'ora locale

A tale proposito notare la sottile differenza tra i due seguenti esempi :

```
var date = new Date("2020-10-13")    // ora 00 di Greenwich

console.log(date.toISOString())      // GMT
2020-10-13T00:00:00.000Z             // ora 00 di Greenwich

console.log(date.toString())         // locale
Tue Oct 13 2020 02:00:00 GMT+0200    // ore 02 locali
```

```
var date = new Date("2020-10-13T00:00:00")  // ora 00 locale
console.log(date.toISOString())      // GMT
2020-10-12T22:00:00.000Z             // ore 22 di Greenwich
console.log(date.toString())         // locale
Tue Oct 13 2020 00:00:00 GMT+0200  // ore 00 locale
```

Riguardo al **±** finale.

- Se omesso, viene richiesto al sistema operativo (come nel secondo caso precedente)
- Se indicato esplicitamente, viene utilizzato per il ricalcolo della data senza richiederlo al sistema operativo.

## Visualizzazione di un oggetto Date

Due sono i metodi principali :

- dataCorrente.**toISOString**()
  Restituisce la data 'nuda e cruda' così come memorizzata all'interno dell'oggetto Date, nel cosiddetto **formato UTC standard** costituito da **24 caratteri** **YYYY-MM-DDThh:mm:ss.mmmZ** con uno **Z** finale che indica che l'ora visualizzata è riferita al meridiano 0 di Greenwich.
  E' l'unico formato accettato dal BSON $date di mongodb per salvare una data all'interno di un DB
  Ad esempio **2022-05-05T11:15:00.000Z**

- dataCorrente.**toString()**
  Restituisce data e ora locali. Rappresenta il metodo di default utilizzato in fase di serializzazione
  La funzione richiede il timezone locale al sistema operative, lo somma alla data contenuta all'interno dell'oggetto Date e la visualizza come data / ora locali. Ad esempio la stessa data precedente verrebbe visualizzata come **2022-05-05T13:15:00.000Z**

## Elenco Completo dei metodi di visualizzazione di una data

```
var dataCorrente = new Date("2020-10-13T17:37:56");
```

| | | |
|---|---|---|
| dataCorrente.**toISOString**() | **2020-10-13T15:37:56.000Z** | // UTC standrad |
| dataCorrente.toISOString().**substr(0,10)** | 2020-10-13 | |
| dataCorrente.**toString()** | Tue Oct 13 2020 **17:37:56** GMT+0200 | // stringa locale |
| dataCorrente.toUTCString() ( | Tue, 13 Oct 2020 15:37:56 GMT | // stringa GMT |
| dataCorrente.toLocaleString() | 13/10/2020, **17:37:56** | // data/ora locali |
| dataCorrente.**toLocaleDateString**() | 13/10/2020 | // solo data locale |
| dataCorrente.toLocaleTimeString('it-IT', { hour12: false }) | , **17:37:56** | // solo ora locale |
| dataCorrente.toDateString() | Tue Oct 13 2020 | // solo data GMT |

offset= dataCorrente.**getTimezoneOffset**() Restituisce l'offset (in minuti) del meridiano corrente (+60 per l'Italia)

## Metodi che restituiscono il timestamp interno

msec=dataCorrente.**getTime( )** Restituisce il **timestamp unix interno** (msec dal 1/1/70)**)**
msec=dataCorrente.**valueOf( )** equivalente al precedente
msec=**Date.now( )** Metodo statico equivalente ai precedenti. Restitusice la data corrente in msec

## Accesso ai singoli campi di una data

ss = dataCorrente.getSeconds ( ) Restituisce i secondi 0-59
mm = dataCorrente.getMinutes( ) Restituisce i minuti 0-59

hh = dataCorrente.getHours( ) Restituisce l'ora da 0-23 riferita al timeZone locale
hh = dataCorrente.getUTCHours( ) Restituisce l'ora da 0-23 calcolata in UTC (1 o 2 ore in meno del precedente)

gg = dataCorrente.**getDate**( ) Restituisce il giorno come numero intero da **1-31** riferita al timeZone locale
gg = dataCorrente.getUTCDate( ) Restituisce il giorno come numero intero da **1-31**espresso in UTC
gg = dataCorrente.getDay ( ) Giorno della settimana 0-6 (Domenica = 0, Lunedì = 1, …… Sabato = 6)
gg = dataCorrente.getUTCDay ( ) Giorno della settimana 0-6 (Domenica = 0, Lunedì = 1, …… Sabato = 6) UTC

me = dataCorrente.getMonth( ) Mese dell'anno 0-11 (attenzione gennaio = 0)
yy = dataCorrente.getFullYear( ) Anno a 4 cifre

## Modifica di una data

I seguenti metodi non restituiscono alcunché, ma provvedono semplicemente a modificare il contenuto di date:
dataCorrente.setTime(val) Si aspetta come parametro i msec complessivi e reimposta l'intera data
dataCorrente.setSeconds (val) Modifica soltanto i secondi
dataCorrente.setMinutes(val) Modifica soltanto i Minuti
dataCorrente.setHours(val) Modifica soltanto l'ora (interpretata come locale)
dataCorrente.setUTCHours(val) Modifica soltanto l'ora (interpretata come UTC)
dataCorrente.setDay(val) Modifica soltanto il giorno della settimana 0-6
dataCorrente.setUTCDay(val) Modifica soltanto il giorno della settimana 0-6 (UTC)
dataCorrente.setMonth(val) mese dell'anno 0-11
dataCorrente.setUTCMonth(val) mese dell'anno 0-11
dataCorrente.setYear(val) Anno a 4 cifre

dataCorrente.**setDate(**val) Modifica soltanto il giorno del mese 1-31. Se come parametro viene passato un numero non previsto nel mese (cioè un numero minore di 1 oppure un numero maggiore di 30 o 31, automaticamente viene aggiornato di conseguenza il numero del **mese** ed eventualmente il numero dell'**anno**.

dataCorrente.setUTCDate(val) Modifica soltanto il giorno del mese 1-31 calcolato come giorno UTC

## Somma, sottrazione e confronto fra date

Poiché il timestamp è un semplice numero intero, **è possibile sommare e sottrarre i timestamp.**

- Il **confronto** può sempre essere eseguito in forma diretta: `if(data1> data2) …`
- Nel caso della **sottrazione** può essere eseguita direttamente sull'oggetto ISODate, **senza** l'utilizzo di .getTime().
- Nel caso della **somma** occorre sempre passare attraverso .getTime()

### 1) Data che si avrà fra 7 giorni.

```
var dataCorrente = new Date()
var nextWeek = dataCorrente.getTime()    +    7 * 24 * 3600 * 1000
alert(new Date(nextWeek))
```

### 2) Differenza fra due date

La differenza fra due oggetti ISODate è un semplice numero indicante i msec di differenza fra le due date.

```
var diff = new Date(_txtN2.value) - new Date(_txtN1.value)
var nGiorni = Math.floor(diff/(24*3600*1000))
```

Se la differenza da valutare è inferiore al mese si può utilizzare la seguente soluzione.

```
var dateDiff = new Date(diff);
var giorni = dateDiff.getUTCDate();  var hours = dateDiff.getUTCHours()
var minutes = dateDiff.getUTCMinutes() // senza UTC verrebbe aggiunto il timeZone
```

**3) Età di una persona**

```
var age=Math.floor((new Date() - dob)/(365*24*3600*1000))
```

**4)** Come impostare una data 'locale' partendo da una data UTC

```
var offset = dataCorrente.getTimezoneOffset() / 60;
var hours = dataCorrente.getHours();
dataCorrente.setHours(hours - offset);
```

**Nota**

Se si imposta una certa data tramite new Date() e poi si vanno a leggere i msec contenuti all'interno della data e li si visualizza all'interno di un time converter, in realtà quello che leggeremo sarò il valore GMT cioè, in inverno, si vede un'ora in meno rispetto all'ora italiana utilizzata nel costruttore (due ore in estate).

## Cenni sulla libreria moment.js

```
src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.1/moment.min.js"
var moment = require('moment');  // nodejs
```

E' un oggetto simile all'oggetto base ISODate di javascript che però al suo interno memorizza più informazioni sulla data memorizzata, in modo da semplificare notevolmente le elaborazioni sulle date, in particolare somme, sottrazioni, differenze mediante i comodissimi metodi **.add()** **.subtract()** e **.diff()**

### Il costruttore moment()

Funzionamento analogo al **`new Date()`** di javascript.
Nel caso di moment() **non** si utilizza il **new** che viene eseguito all'interno della libreria

**`let date = moment()`** restituisce un oggetto moment contenente la data corrente
**`let date=moment("2020-10-13")`** restituisce un oggetto moment riferito alla data **`2020-10-13`**
**`let date=moment("2020-10-13T16:00:00")`** restituisce un oggetto riferito a data e ora indicati.
**`let date=moment("16:00:00", "HH:mm:ss")`** come data viene impostata la data corrente.
Senza il secondo parametro l'istruzione andrebbe in errore.

L'ora viene SEMPRE interpretata come **ora locale** (esattamente come in javascript) a meno che non sia indicata esplicitamente la **Z** finale che indica un Greenwich Time. A differenza però di quanto accade in javascript, se si omette l'ora, essa viene assunta come **ora 0 locale** e non GMT.

Come parametro si possono passare anche i msec (provenienti ad esempio da una differenza).
**In quest'unico caso** il valore viene interpretato come **ora Greenwich** del 1/1/70, che potrà poi essere visualizzata come ora GMT (Greenwich Mean Time) oppure come ora locale.

### La visualizzazione della data come stringa

Gli oggetti moment dispongono del metodo **format()** che per default restituisce **l'ora locale**, cioè richiede al SO il timezone locale e lo aggiunge all'ora GMT.

```
console.log(date.format())         //2020-10-13T17:37:56+02:00
console.log(date.format("h"))  // 17
console.log(date.format("DD/MM/YYYY"))  // 13/10/2020
console.log(date.format("[ore] H [e] m [minuti]"))  // ore 17 e 37 minuti
```

Le espressioni tra parentesi quadre vengono visualizzate così come sono. H=24ore, h=12ore

**Nota:** Il formato restituito da format è estremamente comodo, però senza lo **Z** finale e con lunghezza **> 24**, mongo non lo considera un formato **UTC standard** e pertanto non lo accetta per l'inserimento in un DB.

## I metodi .utc() e .local()

Questi metodi "agiscono" sulla data memorizzata all'interno dell'oggetto moment.
**`.utc()`** fa sì che tutti i successivi metodi format() restituiscano l'ora in GMT (Greenwich Mean Time)
**`.local()`** fa sì che tutti i successivi metodi format() restituiscano l'ora in formato locale.

Le modalità **`utc()`** e **`local()`** possono essere applicate anche direttamente in fase di creazione dell'oggetto moment():

```
let date=moment.utc("2020-10-13T16:00:00")
let date=moment.local("2020-10-13T16:00:00")  // default
```

## Funzioni di estrazione diretta delle informazioni contenute nella data

Su un oggetto moment sono disponibili tutte le seguenti funzioni che ritornano l'informazione richiesta

| **Key** | **Shorthand** |
|---|---|
| years | y |
| quarters | Q |
| months | M  // partendo da 0 ! |
| weeks | w |
| days | d |
| hours | h |
| minutes | m |
| seconds | s |
| milliseconds | ms |

Tutte le chiavi precedenti possono essere scritte indifferentemente in forma singolare oppure plurale (la **`s`** finale è facoltativa)

```
let anno = date.year()
let ora = date.hour()
let newDate = moment().seconds(0)    // azzero i secondi del moment corrente
```

## I metod add() e subtract()

I metodi .add() e .subtract() applicati ad un oggetto moment consentono una rapida elaborazione

```
var date = moment().subtract(1, 'year')
var date = moment().subtract(1, 'day')
var date = moment().add(1, 'day')
```

```
moment().add(7, 'days').add(1, 'months');
moment().add( {days:7, months:1} )
moment().add(7,'days').subtract(1,'months').year(2009).minutes(0).seconds(0)
```

## Il confronto

Le date di moment, così come le ISODate, possono sempre essere confrontate in modo diretto:

```
if(data1 > data2) …
```

## Il metodo diff()

Restitusice la differenza in msec fra due "moment Date" (esattamente come la differenza diretta fra le ISODate). Espone però la possibilità di specificare un secondo parametro che indica come deve essere restituito il risultato. Accetta come parametri gli stessi di cui sopra: `days, hours, minutes, etc.`

// Differenza fra le due date espressa come numero di giorni

```
let ggPermanenza=dataFine.diff(dataInizio, "days")
```

Se si omette il secondo parametro, il risultato viene restituito come numero di millisecondi (intesi come orario di Greemwich del 1/1/70), che possono essere riconvertiti in oggetto moment passandoli come parametro al costruttore moment().

Attenzione però che se si vogliono eseguire delle elaborazioni sul risultato espresso in msec, occorre abilitare la modalità **`utc()`** perché se viene utilizzata la modalità **`local()`**, essa aggiunge una o due ore al numero dei millisecondi:

```
let diff=dataFine.diff(dataInizio)
let ore = moment(diff).utc().hours()  // oppure
let ore = moment.utc(diff).hours()
```

## il metodo toDate() e la conversione in ISODate

Dato un oggetto moment, è possibile convertirlo in ISODate usando il metodo .toDate()

```
var date = moment().subtract(1, 'year')
var jsdate = date.toDate()
```

## I metodi timeTo() e timeFrom()

Consentono di calcolare in modo molto fine la differenza fra due date.
Per l'utilizzo si rimanda alla sezione DOCS del sito ufficiale