OGGETTI & FUNZIONI

OGGETTI

Collezione non ordinata di proprietà/metodi, si possono memorizzare valori attraverso il nome della proprietà

In Javascript gli oggetti possono esistere anche in assenza di classi

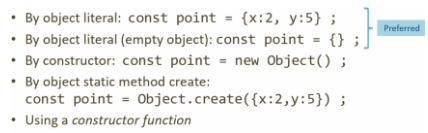
- Sono **dinamici**
  - posso aggiungere/modifica/creare metodi in ogni momento
  - posso aggiungere/modifica/creare proprietà in ogni momento
- non ci sono metodi di controllo degli oggetti
  - proprietà e metodi sono sempre pubblici

Definizione di un oggetto

```
const movie ={
title: 'Titanic',
genre: 'drama',
duration: 200
}
```

Nel caso in cui nel nome di un oggetto ci sono dei valori non permissibili da javascript, bisogna mettere il nome tra virgolette (caratteri speciali/spazio)

Si può anche creare usando le funzioni costruttrici



Accedere ai valori delle proprietà

Si può accedere usando le virgolette o con la dot notation(in assenza di caratteri speciali)

```
console.log(movie['title']);
console.log(movie.title);
```

Aggiunta/eliminazione/modifica proprietà

```
movie.director = 'Cameron';
movie['director']='Cameron';
delete movie.genre;
```

NOTA → Se scrivo movie[title] → title scritto così è una variabile quindi non funzia (tranne se esiste una variabile title che al suo interno abbia il nome di una proprietà)

***NOTA** → se una proprietà non è definita → undefined*

Iterare su oggetti

```
for(const prop in movie){
    console.log(`${prop} is  ${movie[prop]}`);
}
```

```
for( let a in {x: 0, y:3}) {
    console.log(a) ;
}
```

```
x
y
```

Ottenere tutte le chiavi

```
let keys = Object.keys(my_object)
```

Ottenere chiavi-valore

```
let keys_values = Object.entries(my_object)
```

Copia degli oggetti - assign

assign: assegna tutte le proprietà da un oggetto sorgente (movie) ad un oggetto target e restituisce l'oggetto target come risultato

```
const Titanic=Object.assign({}, movie); //shallow copy
```

NOTA: assign si può anche usare per aggiungere proprietà

```
Object.assign(movie, {budget: '200 millions USD'})
```

Unione di oggetti in un nuovo oggetto

```
const improvedMovie= Object.assign({}, movie, {cast: '..'});
```

Verifica di esistenza di proprietà in oggetti

```
console.log('title' in movie);
```

NOTA: LE COPIE SONO SHALLOW

FUNZIONI

Le funzioni possono avere un numero variabile di parametri

PARAMETRI: lista separata di elementi da virgole che possono avere dei valori di default

- Assegna valore di default se non è passato, se non c'è valore di default → undefined

```
function fun (par1, par2, ...arr) { }
```

arr prende tutti gli altri parametri e li mette in un array (deve essere ultimo)

Dichiarazione delle funzioni

- **classico**

```
function do(params) {
  /* do something */
}
```

- **function expression**: definisco una variabile a cui assegno una funzione

```
const fn = function(params) {
  /* do something */
}
```

- **named** function expression

```
const fn = function do(params) {
  /* do something */
```

- **Arrow function**: funzioni usa e getta, infatti molto usate nelle callback

```
const fn = (params) => {
  /* do something */
```

  - Alcune accortezze:
    - Se non ho nessun parametro devo mettere parentesi tonde
    - Se voglio tornare qualcosa devo usare le return (max 1 valore)
    - *Se c'è solo una riga/istruzione, si può omettere parentesi graffe e return*

```
const fun = () => {  /* do something */ }        // no params
const fun = param => {  /* do something */ }     // 1 param
const fun = (param) => {  /* do something */ }   // 1 param
const fun = (par1, par2) => {  /* smtg */ } // 2 params
const fun = (par1 = 1, par2 = 'abc') => {  /* smtg */ }   // default values
```

```
let fourth = (x) => { return square(x)*square(x) ;  }
let fourth = x => square(x)*square(x) ;
```

  - *Capisce in automatico che deve ritornare il risultato dell'unica operazione fatta*

Funzioni annidate

Le funzioni interne **esistono solo all'interno** della madre

- Può accedere alle variabili esterne
- Non è visibile dall'esterno

```
function hypotenuse(a, b) {
    const square = x => x*x ;
    return Math.sqrt(square(a) + square(b));
}

function hypotenuse(a, b) {
    function square(x) { return x*x; }
    return Math.sqrt(square(a) + square(b));
}
```

NOTA: se una funzione è all'interno delle proprietà di un oggetto → metodo

NOTA: se è passata all'interno di un parametro di un oggetto → callback

NOTA: esistono anche:

- **Closure:** funzione interna che può anche essere usata esternamente
- **IIFE** → creo funzione ed eseguo immediatamente.
  - Non posso più richiamarla in futuro

```
( function() {
    let a = 3 ;
    console.log(a) ;
} ) () ;
```

```
let num = ( function() {
    let a = 3 ;
    return a ;
} ) () ;
```

Function construction

Definiscono un tipo di oggetto

- Lettera maiuscola iniziale
- Setta le proprietà con this
- Se avessi scritto duration=0
  - Di default, se assente 0
  - Però dovrebbe essere ultimo

```
'use strict';

function Movie(title, genre, duration, director){
    this.title=title;
    this.duration=duration;
    this.director=director;
    this.genre=genre;
    this.isLong = () =>this.duration > 120;
}


let titanic= new Movie('Titanic', 'drama', 200, 'Cameron');

console.log(`Is titanic long?` + titanic.isLong() );
```

Per creare un'istanza di un oggetto, si usa new

DATE

La maggior parte dei metodi dell'oggetto date funzionano in local time (nel fuso orario che gira sul nostro pc).

La formattazione delle date è quella locale; confronti tra date sono difficili.

- Date ha diversi vincoli, usiamo librerie esterne

DAY.js

Tutti gli oggetti in day.js sono immutabili

```
let now = dayjs()  // today
let date1 = dayjs('2019-12-27T16:00');
        // from ISO 8601 format
let date2 = dayjs('20191227');
        // from 8-digit format
let date3 = dayjs(new Date(2019, 11, 27));
        // from JS Date object
let date5 = dayjs.unix(1530471537);
    // from Unix timestamp

By default, Day.js parses in local time
```

```
console.log(now.format());
        2021-03-02T16:38:38+01:00

console.log(now.format('YYYY-MM [on the] DD'));
        2021-03 on the 02

console.log(now.toString());
        Tue, 02 Mar 2021 15:43:46 GMT

By default, Day.js displays in local time
```

```
# obj.unit() -> get
# obj.unit(new_val) -> set

let now2 = now.date(15);
let now2 = now.set('date', 15);
        2021-03-15T16:50:26+01:00

let now3 = now.minute(45);
let now3 = now.set('minute',45);
        2021-03-02T16:45:26+01:00

let today_day = now.day();
let today_day = now.get('day');
```

Unit	Shorthand	Description
date	D	Date of Month
day	d	Day of Week (Sunday as 0, Saturday as 6)
month	M	Month (January as 0, December as 11)
year	y	Year
hour	h	Hour
minute	m	Minute
second	s	Second
millisecond	ms	Millisecond

```
let wow = dayjs('2019-01-25').add(1, 'day').subtract(1, 'year').year(2009).toString() ;
// "Sun, 25 Jan 2009 23:00:00 GMT"
```

- Methods to "modify" a date (and return a modified one)
- `.add` / `.subtract`
- `.startOf` / `.endOf`
- `d1.diff(d2, 'unit')`
- Specify the unit to be added/subtracted/rounded
- Can be easily *chained*

- Day.js objects can be compared
- `.isBefore` / `.isSame` / `.isAfter`
- `.isBetween`
- `.isLeapYear` / `.daysInMonth`

Esistono **plugin esterni** che si possono aggiungere:

- Anno bisestile
- .isBetween
- .isBefore .isSame .isAfter
- .daysInMonth

```
const isLeapYear =
   require('dayjs/plugin/isLeapYear') ;
        // load plugin

dayjs.extend(isLeapYear) ;
        // register plugin

console.log(now.isLeapYear()) ;
        // use function
```

```
npm install dayjs
```

```
> npm init
```

Metodi per modificare una data

- .add
- .substract
- .startOf
- .endOf
- d1.diff(d2, 'unit')

```
let wow = dayjs('2019-01-25').add(1, 'day').subtract(1, 'year').year(2009).toString() ;
// "Sun, 25 Jan 2009 23:00:00 GMT"
```

esempio

```
'use strict';

// CommonJS
const dayjs= require('dayjs');  7.2k (gzipped: 3k)

let oggi=dayjs();
console.log(oggi);
```

→


```
'$L': 'en',
'$d': 2024-03-12T08:08:21.292Z,
'$y': 2024,
'$M': 2,
'$D': 12,
'$W': 2,
'$H': 9,
'$m': 8,
'$s': 21,
'$ms': 292,
'$x': {},
'$isDayjsObject': true
```


Con common js → file.js

Nota: l'orario è quello inglese (1 ora in meno rispetto ad italia; $M è due perché parte da 0)

```
console.log(oggi.format('YYYY-MM-DD'));
```

→ `2024-03-12`

Nota: con ES module → .mjs e non serve specificare 'use strict';