Corso Enaip
Rimini 03/03/2023

Agenda

Gli Oggetti in Javascript

Oggetti in JavaScript

Un **oggetto** è un **contenitore** di valori eterogenei, messi insieme a formare una struttura dati unica e tale da avere generalmente una particolare identità. Normalmente, infatti, un oggetto è utilizzato per rappresentare un'entità specifica come ad esempio una persona, un ordine, una fattura, una prenotazione, etc. tramite un'aggregazione di dati e di funzionalità.

Oggetti in Javascript

E’ un **tipo di dato** che contiene **gruppi di dati (proprietà)**, identificati da un **nome**.

Un oggetto tipicamente possiede:

- Dati: detti **proprietà** e rappresentati da coppie di **nome-valore**;
- Funzionalità: sono dette **metodi** e sono rappresentate da **funzioni**;

Oggetti in Javascript

Un oggetto può essere creato tramite le parentesi graffe **{}** con un opzionale lista di proprietà**,** oppure attraverso l'utilizzo di un costruttore ad. esempio **new Object()**.

```
var oggetto = new Object(); // sintassi "costruttore oggetto"
var oggetto = {};  // sintassi "oggetto letterale"
```

Proprietà

Proprietà di un oggetto

Una proprietà **è una coppia “chiave: valore”**, dove la “chiave” è una stringa (conosciuta anche come “nome di proprietà” o “identificatore”), mentre “valore” può essere qualsiasi cosa (ad esempio: una stringa, un boolean, un numero, un array oppure un oggetto).

Proprietà di un oggetto

Una proprietà ha una **chiave** (conosciuta anche come “**nome**” o “identificatore”) prima dei due punti **":"**, ed un valore alla sua destra. Per i nomi delle proprietà di un oggetto non abbiamo le restrizioni dei nomi delle variabili.

```
var utente = {     // un oggetto
    nome: "Francesco",  // una chiave "nome" memorizza il valore "Francesco"
    cognome: "Totti",   // una chiave "cognome" memorizza il valore "Totti"
    eta: 45, // una chiave "eta" memorizza il valore 30
};
```

Proprietà di un oggetto

Nell’oggetto **utente** ci sono **tre** proprietà:

La prima proprietà ha come nome "nome" e come valore "Francesco".

La seconda proprietà ha come nome "cognome" e come valore "Totti".

La terza ha come nome "eta" e come valore 45.

ATTENZIONE

Abbiamo detto che il nome di una proprietà non ha nessuna restrizione, ma cosa succede se usiamo uno dei caratteri non validi nella definizione di una variabile? Ad esempio il “-”? Dobbiamo utilizzare **ad utilizzare i doppi apici**.

```
var player = {     // un oggetto
    "first-name": "Francesco",  // una chiave "nome" memorizza il valore "Francesco"
    "last-name": "Totti",   // una chiave "cognome" memorizza il valore "Totti"
    "age": 45, // una chiave "eta" memorizza il valore 30
};
```

Accedere alle proprietà

Per accedere ai valori memorizzati in una proprietà di un oggetto abbiamo due approcci.

1) dot-notation (la notazione puntata)
2) utilizzando la stringa che identifica il nome della proprietà.

Accedere alle proprietà

```
//indicando l'oggetto e la proprietà a cui siamo interessati separandoli con un punto
var nome = utente.nome;

//indicando l'oggetto e il nome della proprietà sotto forma di stringa tra parentesi quadre
var firstName = player["first-name"];
```

Accedere alle proprietà

Il tentativo di accedere ad una proprietà **non definita** in un oggetto **non genera un errore**, ma restituisce il valore **undefined**. Nell'esempio riportato nella prossima slide, quindi, la variabile team avrà valore undefined.

Accedere alle proprietà

Il tentativo di accedere ad una proprietà **non definita** in un oggetto **non genera un errore**, ma restituisce il valore **undefined**. Nell'esempio riportato qui sotto, quindi, la variabile **team** avrà valore undefined.

```
var player = {     // un oggetto
    "first-name": "Francesco",  // una chiave "nome" memorizza il valore "Francesco"
    "last-name": "Totti",   // una chiave "cognome" memorizza il valore "Totti"
    "age": 45, // una chiave "eta" memorizza il valore 30
};

var team = player.team; //la variabile team avrà come valore "undefined"
```

Creazione di una proprietà per definizione

Se assegniamo un valore ad una proprietà non definita creiamo di fatto questa proprietà inizializzandola con il valore assegnato.

```
var player = {     // un oggetto
    "first-name": "Francesco",  // una chiave "nome" memorizza il valore "Francesco"
    "last-name": "Totti",   // una chiave "cognome" memorizza il valore "Totti"
    "age": 45, // una chiave "eta" memorizza il valore 30
};

player.team = "As Roma 1927";
```

Proprietà calcolate

Possiamo utilizzare le parentesi quadre al momento della creazione di un oggetto letterale. Questo metodo viene chiamato **calcolo delle proprietà**.

```
var nomeGiocatore = "Aldair";
var player = {
  [nomeGiocatore]: 6, // il nome della proprietà viene preso dalla variabile nomeGiocatore
};
console.log( player.Aldair ); // 6 se nomeGiocatore="Aldair"
```

Proprietà calcolate

Possiamo utilizzare anche espressioni più complesse all'interno delle parentesi quadre.

```
var nomeGiocatore = "Aldair";
var player = {
  ['numeroMaglia' + nomeGiocatore]: 6, // il nome della proprietà viene preso dalla variabile nomeGiocatore
};
console.log( player.numeroMagliaAldair ); // 6 numeroMagliaAldair
```

Abbreviazione per il valore di una proprietà

Possiamo usare delle variabili esistenti come valori per i nomi delle proprietà.

```
function makePlayer(nome, cognome, eta) {
    return {
      nome: nome,
      cognome: cognome,
      eta: eta
      // ...altre proprietà
    };
  }

var player = makePlayer("Francesco", "Totti", 45);
console.log(player.nome); // Francesco
```

Abbreviazione per il valore di una proprietà

Nell’esempio precedente, le proprietà hanno lo stesso nome delle variabili. Il caso d’uso di creare una proprietà da una variabile è molto frequente, tanto che, per comodità, esiste una speciale abbreviazione.

Abbreviazione per il valore di una proprietà

```
function makePlayer(nome, cognome, eta) {
    return {
      nome,
      cognome,
      eta: eta
      // ...altre proprietà
    };
  }

var player = makePlayer("Francesco", "Totti", 45);
console.log(player.nome); // Francesco
```

Controllo di esistenza

Un'importante caratteristica degli oggetti, in Javascript, è che **è possibile accedere a una qualsiasi proprietà**. Non ci sarà alcun errore se la proprietà non esiste. L'accesso ad una proprietà non esistente ritornerà **undefined**.

```
var player = {};

console.log( player.nome === undefined ); // true significa "properietà non esistente"
```

Controllo di esistenza, operatore “in”

Esiste anche uno speciale operatore **"in"** per lo stesso scopo.

```
var player = {nome: "Francesco", eta: 30};

console.log( "nome" in player ); // true, significa che player.nome esiste
console.log( "cognome" in player ); // false, significa che player.cognome non esiste
```

Attenzione!

Il confronto stretto con **"=== undefined"** funziona correttamente. Ma c'è un particolare caso in cui questo controllo fallisce, mentre con l'operatore **"in"** funziona correttamente. Questo accade quando una proprietà esiste, ma contiene **undefined**.

```
var player = {nome: "Francesco", cognome: undefined, eta: 30};

console.log( "nome" in player ); // true, significa che player.nome esiste
console.log( "cognome" in player ); // false, significa che player.cognome non esiste
console.log( player.cognome ); // undefined, quindi -- non esiste la proprietà?
```

Il ciclo “for…in”

Per attraversare tutte le chiavi di un oggetto, esiste una speciale forma di ciclo: il **for..in**.

```
//sintassi
for (chiave in oggetto) {
    // esegue il corpo del ciclo per ogni proprietà dell'oggetto
}

var player = {nome: "Francesco", cognome: "Totti", eta: 45};
//esempio
for (var key in player) {
    // keys
    console.log( player );  // nome, cognome, eta
    // valori delle keys
    console.log( player[key] ); // Francesco, Totti, 45
}
```

L’ordine delle proprietà?

Gli oggetti sono ordinati? Se iteriamo un oggetto otterremo le sue proprietà nello stesso ordine in cui le abbiamo aggiunte?

Una risposta breve è: **“sono ordinati in modo speciale”**. Le proprietà che hanno **numeri interi come chiavi** vengono ordinate, le altre appaiono seguendo **l’ordine di creazione**.

L’ordine delle proprietà?

```
var classifica = {
  "1": "Roma",
  "6": "Juventus",
  "4": "Inter",
  // ..,
  "20": "Lazio"
};

for (var posto in classifica) {
  alert(posto); // 1, 4, 6, 20
}

var player = {nome: "Francesco", cognome: "Totti"};
player.eta = 45; // aggiungiamone un'altra
for (var key in player) {
  alert(key); // nome, cognome, eta
}
```

Cancellare una proprietà?

Per rimuovere una proprietà, possiamo utilizzare l'operatore **delete**.

```
var player = {nome: "Francesco", cognome: "Totti", eta: 45};
delete player.cognome;
console.log( "cognome" in player ); //false, significa che player.cognome è stata cancellata
```

Domande?

Riepilogo

Gli oggetti sono **arrays associativi** con diverse caratteristiche speciali:

Possono memorizzare proprietà (coppie di chiave-valore) in cui:

- Il nome della proprietà (chiave) deve essere composta da una o più stringhe o simboli (solitamente stringhe).
- I valori possono essere di qualsiasi tipo

Per accedere ad una proprietà possiamo utilizzare:

- La notazione puntata: obj.property.
- La notazione con parentesi quadre obj["property"]. Questa notazione consente di accettare chiavi dalle variabili, come obj[varWithKey].

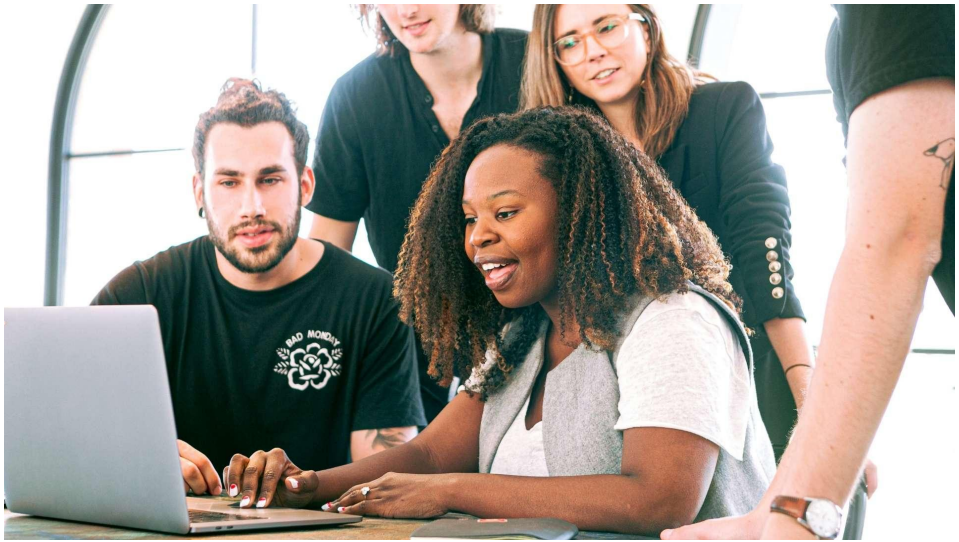
Operatori specifici:

- Per cancellare una proprietà: delete obj.prop.
- Per controllare se un una proprietà con un certo nome esiste: "key" in obj.
- Per iterare un oggetto: for(let key in obj).

Esercitazione 1

Scrivi il seguente codice, una riga per ogni azione:

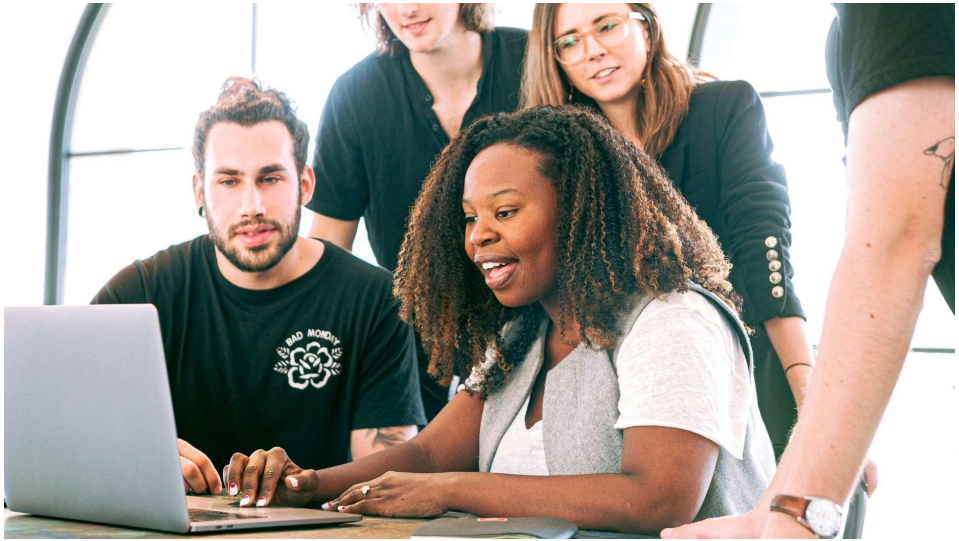
1. Crea un oggetto vuoto **Calciatore**.
2. Aggiungi la proprietà **nome** con valore **Lorenzo**.
3. Aggiungi la proprietà cognome con valore **Pellegrini**.
4. Cambia il valore di **nome** con **Giuseppe**.
5. Rimuovi la proprietà **cognome** dall'oggetto.



AdriasOnline
Connessi per accogliere

Esercitazione 2

Scrivi la funzione **isEmpty(obj)** che ritorna **true** se l’oggetto non ha proprietà, altrimenti ritorna false.

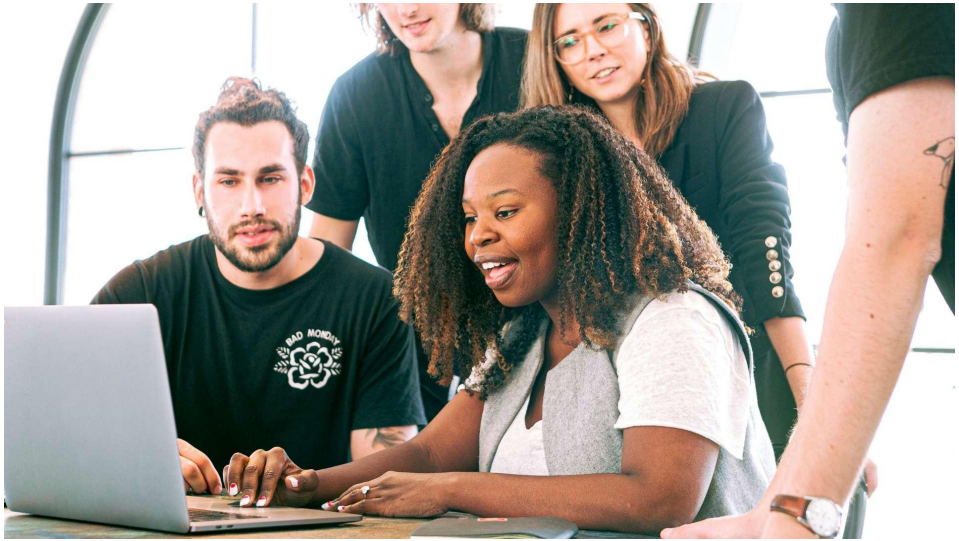


AdriasOnline
Connessi per accogliere

Esercitazione 3

Scrivi una funzione che stampi il nome e il cognome dei calciatori che hanno il numero della maglia uguale a 10.

La lista dei giocatori è definita in questo file https://github.com/Adrias-Online/corso-ifts/blob/main/esercizi-lezione-03/calciatori.js



AdriasOnline
Connessi per accogliere

Metodi

I metodi degli oggetti

Come abbiamo già detto gli oggetti sono utilizzati per **rappresentare dei dati** (entità) del mondo reale come *Utenti*, *Giocatori*, *Ordini* e etc.. queste entità possono effettuare delle **azioni**. Ad esempio un Utente può effettuare un acquisto, commentare un post e etc.. **Queste azioni in Javascript vengono chiamate metodi**. Un metodo è rappresentato da una **funzione**.

I metodi degli oggetti

```
var player = {nome: "Francesco", cognome: "Totti"};
player.esulta = function(){
    console.log("Gooooal!");
};
player.esulta(); //Viene stampato sulla console: Gooooal!
```

I metodi degli oggetti

```
var player = {nome: "Francesco", cognome: "Totti"};
var esulta = function(){
    console.log("Gooooal!");
};
player.esulta = esulta;
player.esulta(); //Viene stampato sulla console: Gooooal!
```

I metodi degli oggetti

Esiste una sintassi più breve per i metodi in un oggetto letterale

```
player = {
   esulta: function() {
       console.log("Gooooal!");
   }
};

// la sintassi più breve
player = {
   esulta() { // equivalente a "esulta: function(){...}"
       console.log("Gooooal!");
   }
};
```

this

E’ molto comune che, per eseguire determinate azioni, un metodo abbia necessità di **accedere alle proprietà** dell’oggetto.

Ad esempio, il codice dentro **player.esulta()** potrebbe aver bisogno del nome del player.

Per accedere all’oggetto un metodo può utilizzare la parola chiave **this**.

this

```
var player = {
  nome: "Francesco",
  esulta() {
      console.log(this.nome +": Gooooal!" ); // Francesco: Gooooal!"
  }
};
```

this

In JavaScript, la parola chiave **“this”** si comporta diversamente da come fa in molti altri linguaggi di programmazione. Essa può essere usata **in qualsiasi funzione**, anche se non si tratta del **metodo di un oggetto**.

```
//Questo codice non produce nessun errore.
function esulta() {
  console.log( this.nome );
}
```

this

Il “this” viene interpretato dal motore Javascript in modo implicito a seconda dei casi. Ad esempio nel codice sottostante ad ogni invocazione del metodo f, il this accede alle proprietà degli oggetti corrispondenti.

```
var player = {nome: "Francesco"}
var professore = {nome: "Giuseppe"}
var fn = function(){console.log(this.nome);};
player.f = fn;
professore.f = fn;
player.f(); //Francesco
professore.f(); //Giuseppe
professore["f"](); //Giuseppe
```

Domande?

Riepilogo

Le funzioni che vengono memorizzate come proprietà di un oggetto vengono dette “metodi”.

- I metodi consentono agli oggetti di “agire”, **come object.doSomething().**
- I metodi possono riferirsi all’oggetto tramite **this**.
- Il valore **this** viene definito durante l’esecuzione (run-time).

Quando una funzione viene dichiarata, può utilizzare **this**, ma questo **this** non avrà alcun valore fino a che la funzione non verrà chiamata.

- Una funzione può essere copiata in vari oggetti.
- Quando una funzione viene chiamata come “metodo”: object.method(), il valore di this durante la chiamata si riferisce a object.

Esercitazione 1

Scrivi un programma per la gestione di un garage.

Definisci un oggetto che rappresenti un automobile, dovrà contenere almeno marca del veicolo e nome del modello.

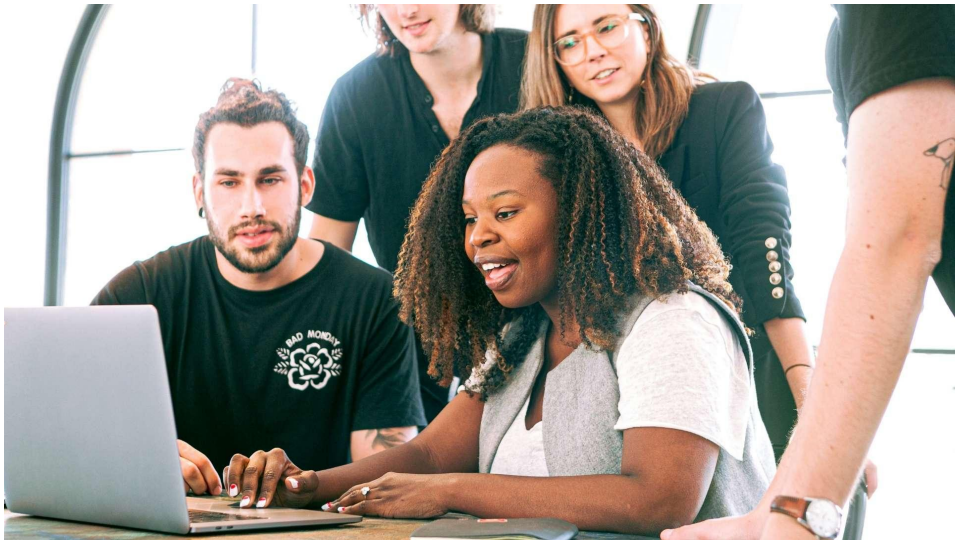
Definisci un oggetto che rappresenti il garage.

Scrivi una funzione che inserisca un veicolo nel garage.

Scrivi una funzione che prenda in input una marca e stampi i modelli presenti nel garage di quella stessa marca.

Scrivi una funzione che rimuove un veicolo dal garage.

Puoi usare un array come base per salvare le automobili.



Prossima lezione

Gli Oggetti definiti con il costruttore
Le Clousure e scope
Le funzioni lambda o arrows function