JAVASCRIPT

Gli oggetti

Oggetto

L’oggetto è una variabile. Una variabile specifica più complessa di quelle viste finora. Ogni oggetto è composto da una o più proprietà, ed ogni proprietà ha il suo valore. Ad esempio il modo per creare un semplice oggetto è :

var automobile = {marca :" Fiat", modello : "500", colore: " Rosso"}

Prime osservazioni

Possiamo subito notare che un oggetto non è altro che una variabile con delle proprietà ognuna separata da una virgola.

E ad ogni proprietà viene assegnato un valore tramite i due punti : il tutto racchiuso da parentesi graffe e terminate con il solito punto e virgola.

Per accedere ad una proprietà di un oggetto è sufficiente richiamare il nome della variabile e tramite l'operatore dot (**il punto**) posso «entrare» nell'oggetto e richiamare la proprietà. Nel nostro esempio se voglio stampare a video la marca dell'oggetto automobile posso scrivere:

```
document.write(automobile.marca);
```

Un altro modo alternativo per dichiarare ed istanziare un oggetto è nel seguente modo :

```
var automobile = new Object();
automobile.marca = ‘’Fiat’’;
automobile.modello = ‘’500’’;
automobile.colore = ‘’Rosso’’;

Document.write(automobile.marca);
```

Oggetti Innestati

E’ possibile avere l’esigenza di inserire un oggetto all’interno di un altro oggetto, in altre parole di innestare un oggetto all’interno di un altro. E’ possibile farlo tramite la seguente sintassi :

```
var persona = {
        nome: 'Mario',
        cognome: 'Rossi',
        eta: 30,
        indirizzo: {
                via: 'Via casilina',
                civico: 23
        }
}
```

Per richiamare una proprietà di un oggetto innestato è sufficiente usare sempre l'operatore dot partendo sempre dalla classe principale :

```
document.write(persona.indirizzo.via);
```

E se invece di un indirizzo voglio dare la possibilità di inserire due o più indirizzi? Posso usare un array come già sappiamo :

```
var persona = {
        nome: ‘’Mario’’,
        cognome: ‘’Rossi’’,
        eta: 30,
        indirizzo: [{
                via: ‘’Via casilina’’,
                civico: 23
        },{
                via: ‘’Via prenestina’’,
                civico: 110
        }]
}

document.write(persona.indirizzo[1].via);
```

Funzioni nell'oggetto

Naturalmente come ogni linguaggio di programmazione, gli oggetti, oltre ad avere le proprietà, hanno anche le funzioni.

Per scrivere una funzione all'interno di un oggetto è sufficiente dare un nome alla funzione seguito dai : e dalla parola chiave function seguita dalle parentesi tonde () :

```
var persona = {
        nome: 'Mario’,
        cognome: 'Rossi',
        eta: 30,
        indirizzo: [{
                via: 'Via casilina',
                civico: 23
        },{
                via: ‘’Via prenestina’’,
                civico: 110
        }],

        stampa: function(nome,cognome){
                document.write(nome + ' ' + cognome);
        }
}

persona.stampa("Riccardo","Cattaneo");
```

Array Associativi

Vediamo un altro esempio di oggetto.

```
let book = {
    "main title": "JavaScript",
    "sub-title": "The Definitive Guide",
    for: "all audiences",
    author: {
            firstname: "David",
            surname: "Flanagan"
            }
    };
```

```
let book={
    "main title":"Libro Javascript",
    "sub-title":"Linguaggio potente",
    for:"Developer",
    author:{
        firtname:"Rob",
        lastname:"Del"
    }
}
```

Possiamo notare che alcuni attributo dell'oggetto sono con spazi e tra apici. In questo caso per richiamarli dobbiamo usare la seguente istruzione

```
book["main title"]
```

Array Associativi

Per spiegare il caso precedente facciamo un ulteriore esempio

```
object.property
object["property"]
```

La prima sintassi, che utilizza il punto e un identificatore, è simile alla sintassi utilizzata per accedere a un campo statico di una struct o di un oggetto in C o Java.

La seconda sintassi, che usa le parentesi quadre e una stringa, assomiglia all'accesso a un array, ma a un array indicizzato da stringhe anziché da numeri.

Questo tipo di array è noto come array associativo (o hash o mappa o dizionario).

Gli oggetti JavaScript sono array associativi.

Modello a oggetti

JavaScript è un linguaggio ad oggetti basato su **prototipi**, piuttosto che sulle classi. A causa di questa diversa base, può essere meno evidente come JavaScript permette di creare gerarchie di oggetti e di avere l'ereditarietà delle proprietà e dei loro valori.

I linguaggi ad oggetti basati su classi, come **Java**, si basano su due entità distinte: le **classi** e le **istanze**.

Una classe definisce tutte le proprietà che caratterizzano una determinata collezione di oggetti. Una classe è un'entità astratta, più che un membro del gruppo di oggetti che descrive. Per esempio, la classe Impiegato può rappresentare il gruppo di tutti i dipendenti.

Un'istanza, d'altra parte, è l'istanziazione di una classe; cioè uno dei suoi membri. Per esempio Luca può essere un'istanza della classe Impiegato, che rappresenta un particolare individuo come un dipendente. Un'istanza ha esattamente le stesse proprietà della classe a cui appartiene.

Un linguaggio basato su prototipi, come JavaScript, non fa questa distinzione: **ha solo oggetti**. Introduce la nozione di oggetto prototipo (prototypical object), un oggetto usato come modello da cui prendere le proprietà iniziali per un nuovo oggetto.

Ogni oggetto può specificare le sue proprietà, anche quando viene creato o in fase di esecuzione. Inoltre, ogni oggetto può essere associato ad un altro oggetto come prototipo, consentendo al secondo oggetto di condividere le proprietà del primo.

Nei linguaggi basati su classi, le classi vengono definite in classi separate. In queste definizioni è possibile specificare metodi speciali, chiamati costruttori, per creare istanze della classe. Un costruttore può specificare i valori iniziali per le proprietà dell'istanza ed eseguire altre elaborazioni adatte al momento della creazione. Per creare le istanze di una classe si utilizza l'operatore new associato al metodo costruttore.

JavaScript segue un modello simile, ma **non prevede la definizione della classe separata dal costruttore**. Invece, per creare oggetti con un particolare set di proprietà e valori si definisce una **funzione costruttore**. **Ogni funzione JavaScript può essere usata come costruttore**. Per creare un nuovo oggetto si utilizza l'operatore new associato a una funzione costruttore.

Costruttore

In Javascript non c'è bisogno di definire una classe ma definiamo direttamente l'oggetto così come ci serve al momento ed eventualmente possiamo modificare la sua struttura nel corso dell'esecuzione del nostro script.

Immaginiamo però di aver bisogno di più oggetti dello stesso tipo, ad esempio di più oggetti persona, che condividono la stessa struttura:

```
var persona = {
        nome: "Mario",
        cognome: "Rossi",
        indirizzo: "Via Garibaldi, 50 - Roma",
        email: "mario.rossi@html.it",
        mostraNomeCompleto: function() { ... },
        calcolaCodiceFiscale: function() { ... }
}
```

Utilizzando la **notazione letterale** saremmo costretti a ripetere la definizione per ciascun oggetto che vogliamo creare. In altre parole, ricorrendo alla notazione letterale nella definizione degli oggetti otteniamo un risultato non riutilizzabile.

Per evitare quindi di dover ridefinire da zero oggetti che hanno la stessa struttura possiamo ricorrere ad un costruttore. Un costruttore non è altro che una normale funzione JavaScript invocata mediante l'operatore new. Vediamo ad esempio come creare un costruttore per l'oggetto persona:

```
function persona() {
        this.nome = "";
        this.cognome = "";
        this.indirizzo = "";
        this.email = "";
        this.mostraNomeCompleto = function() {...};
        this.calcolaCodiceFiscale = function() {...};
}
```

Questa funzione definisce le proprietà del nostro oggetto assegnandole a se stessa (**this**) in qualità di oggetto ed impostando i valori predefiniti. Per creare un oggetto di tipo persona dovremo a questo punto invocare la funzione premettendo l'operatore **new**:

```
var marioRossi = new persona();
marioRossi.nome = "Mario";
marioRossi.cognome = "Rossi";

var giuseppeVerdi = new persona();
giuseppeVerdi.nome = "Giuseppe";
giuseppeVerdi.cognome = "Verdi";
```

In questo modo nella creazione di più oggetti con la stessa struttura ci limiteremo ad impostare i soli valori specifici che differenziano un oggetto dall'altro.

Nella definizione di un costruttore possiamo prevedere la presenza di **parametri** che possiamo utilizzare nell'inizializzazione del nostro oggetto. Consideriamo ad esempio la seguente definizione del costruttore dell'oggetto persona:

```
function persona(nome, cognome) {
        this.nome = nome;
        this.cognome = cognome;
        this.indirizzo = "";
        this.email = "";
        this.mostraNomeCompleto = function() {...};
        this.calcolaCodiceFiscale = function() {...};
}
```

Esso ci consente di creare ed inizializzare un oggetto specificando i valori nella chiamata al costruttore:

```
var marioRossi = new persona("Mario", "Rossi");
var giuseppeVerdi = new persona("Giuseppe", "Verdi");
```

È fondamentale utilizzare l'operatore **new** nella creazione di un oggetto tramite costruttore. Infatti se lo omettiamo, magari per dimenticanza, quello che otterremo non sarà la creazione di un oggetto ma l'esecuzione della funzione, con risultati imprevedibili. Ad esempio, se tentiamo di creare un oggetto persona omettendo l'operatore new:

```
var marioRossi = persona();
```

il valore della variabile marioRossi sarà **undefined**, dal momento che la funzione persona() non restituisce alcun valore.

Un altro modo per creare oggetti è utilizzare le classi. Una classe funge da "progetto" per creare oggetti.

```
class Utente {
  constructor(nome, eta) {
    this.nome = nome;
    this.eta = eta;
  }
  saluta() {
    console.log('Ciao! ‘ + this.nome);
  }
}
const utenteUno = new Utente('Manuel', 35);
```

Qui, `utenteUno` è un'istanza della classe `Utente` e ha accesso al metodo `saluta`.

In questo corso, non ci concentreremo troppo sull'uso delle classi, ma è un concetto utile da conoscere. Ora possiamo passare al prossimo argomento.

Prototype

La flessibilità degli oggetti JavaScript si esprime principalmente nella possibilità di **modificare la struttura anche dopo la creazione**. Anche utilizzando un costruttore per creare un oggetto, continuiamo a disporre di questa possibilità.

Riprendendo il costruttore creato nelle lezioni precedenti (persona), possiamo ad esempio scrivere il seguente codice:

```
var marioRossi = new persona("Mario", "Rossi");
var giuseppeVerdi = new persona("Giuseppe", "Verdi");

marioRossi.telefono = "0612345678";
```

Creando una nuova proprietà telefono per l'oggetto Mario Rossi, senza influenzare la struttura di Giuseppe Verdi. In sostanza, nella creazione di oggetti possiamo partire da una struttura comune definita da un costruttore per poi personalizzarla in base alle nostre esigenze.

Ma come fare per modificare la struttura di tutti gli oggetti creati tramite un costruttore? Se ad esempio, dopo aver creato diversi oggetti dal costruttore persona() vogliamo aggiungere per tutti la proprietà telefono, possiamo sfruttare una delle caratteristiche più interessanti della programmazione ad oggetti di JavaScript: il **prototype**. Nel nostro caso procederemo nel seguente modo:

```
persona.prototype.telefono = "123456";
```

Questo assegnamento fa sì che **tutti gli oggetti creati tramite il costruttore persona()** abbiano istantaneamente tra le loro proprietà anche la proprietà telefono valorizzata al valore indicato.

Ad essere precisi, **la nuova proprietà non è direttamente agganciata a ciascun oggetto, ma accessibile come se fosse una sua proprietà**. Questo grazie al meccanismo di prototyping che sta alla base dell'ereditarietà nella programmazione ad oggetti in JavaScript.

Metodi apply() e call()

JavaScript, come già sappiamo, considera tutto come un oggetto e quindi rappresenta in questo modo anche le funzioni (e quindi i metodi). Ciascuna funzione presenta infatti proprietà e metodi proprio come un normale oggetto. Il prototipo dell'oggetto Function presenta due metodi di fondamentale importanza che sono **apply()** e **call()**.

Il loro comportamento è simile, presentano una semplice differenza nella gestione dei parametri. Essi permettono infatti di invocare una determinata funzione definendo quale sarà il suo scope passandoglielo come primo parametro. È possibile inoltre fornire ulteriori parametri che verranno in qualche modo "passati" alla funzione oggetto dell'invocazione.

La differenza tra apply e call è proprio questa: apply() oltre all'oggetto scope accetta un **array di parametri**, mentre call() accetta un **numero indefinito di parametri**. Passiamo all'esempio:

```
function myFunction(a, b) {
    return a * b;
}
myObject = myFunction.call(myObject, 10, 2);
```

```
function myFunction(a, b) {
    return a * b;
}
myArray = [10, 2];
myObject = myFunction.apply(myObject, myArray);
```

Esercizio 3.1

Progettare l'oggetto canzoni che contiene delle proprietà: canzone1, canzone2, ecc… Queste proprietà a sua volta sono degli oggetti che hanno altre proprietà: titolo, nomeCantante e anno. Inserire poi alcuni dati a piacere e visualizzarli. Dopo fare inserire all'utente delle nuove canzoni attraverso il prompt dei comandi. Visualizzare il nuovo oggetto così costruito.

Esercizio 3.2

Progettare un oggetto rubrica che ha come proprietà gli utenti. Dopo, per ogni utente specificare altre proprietà: utente, nome, cognome, telefono e indirizzo. Dove indirizzo è a sua volta un altro oggetto contenente altre proprietà.

Quindi popolare la rubrica con dei dati a piacere. Dopo eliminare l'ultimo elemento e visualizzare nuovamente la rubrica così ottenuta.

Esercizio 3.3

1. Definisci un oggetto "studente" con le seguenti proprietà:

- nome: una stringa con il nome dello studente
- cognome: una stringa con il cognome dello studente
- voto: un numero che rappresenta il voto dello studente

2. Aggiungi un metodo all'oggetto "studente" chiamato "calcolaMedia" che calcola la media di due voti passati come argomenti e restituisce il risultato.

3. Crea un nuovo oggetto "studente1" utilizzando l'oggetto "studente" come modello. Assegna valori alle proprietà "nome", "cognome" e "voto".

4. Chiama il metodo "calcolaMedia" sull'oggetto "studente1" passando due voti come argomenti. Stampa il risultato.

5. Crea un altro oggetto "studente2" utilizzando l'oggetto "studente" come modello. Assegna valori alle proprietà "nome", "cognome" e "voto".

6. Chiama il metodo "calcolaMedia" sull'oggetto "studente2" passando due voti come argomenti. Stampa il risultato.

7. Confronta i voti dei due studenti ("studente1" e "studente2") e stampa un messaggio che indica quale studente ha ottenuto un voto più alto.

8. Modifica il voto dello "studente1" e del "studente2" assegnando nuovi valori alla proprietà "voto".

9. Chiama nuovamente il metodo "calcolaMedia" sui due studenti con i nuovi voti. Stampa i nuovi risultati.

10. Confronta i nuovi voti dei due studenti e stampa un messaggio che indica quale studente ha ottenuto un voto più alto.