Array e oggetti

Corso di Laboratorio di Informatica

Prof. Dei Rossi, Leonardo Essam

Gli array in JS (1)

In JavaScript, un array rappresenta una struttura dati indicizzata che consente di organizzare in sequenza un insieme di valori.

Questa collezione ordinata permette di gestire, attraverso un unico identificatore[1], più elementi potenzialmente eterogenei (stringhe, numeri, oggetti…), rendendo più agevole l'elaborazione di insiemi di dati.

La sintassi più comune per dichiarare un array è utilizzando le parentesi quadre:

```
1 let colori = ["rosso", "verde", "blu"];
```

[1] Il nome della variabile.

Gli array in JS (2)

Gli array in JavaScript adottano un indice numerico basato su zero:

- `colori[0]` restituisce `"rosso"`;
- `colori[1]` restituisce `"verde"`;
- `colori[2]` restituisce `"blu"`.

È possibile accedere anche in scrittura a ciascun elemento:

```
1  colori[1] = "giallo"; // "verde" ==> "giallo"
```

Gli array in JS (3)

Ogni array espone la proprietà `.length`, che consente di ottenere in tempo costante il numero di elementi presenti.

Questa informazione è molto importante per iterare correttamente sull'intera sequenza:

```
1  let numeri = [1, 2, 3, 4];
2  console.log(numeri.length); // 4
```

Iterare un array (1)

In JavaScript, esistono tre metodi per iterare un array:

- Ciclo `for` "classico" basato sull'indice;
- Ciclo `for ... of`, che semplifica la lettura sequenziale dei valori;
- Metodo `.forEach()`, che applica una funzione f a ciascun elemento.

Iterare un array (2)

L'iterazione tramite ciclo segue la seguente sintassi:

```
let colori = ["rosso", "verde", "blu"];

for (let i = 0; i < colori.length; i++) {
    let colore = colori[i];
    console.log(colore);
}
```

Iterare un array (3)

In caso si volesse iterare un array senza tenere in considerazione l'indice i, si può usare la seguente scrittura:

```
1  let array = ["a", "b", "c", "d"];
2
3  for (let element of array) {
4      console.log(element);
5  }
```

Iterare un array (4)

Quando vi è invece la necessità di applicare una funzione f su tutti gli elementi dell'array, è possibile usare la seguente scrittura:

```
let array = ["a", "b", "c", "d"];
function stampa(n) {
    console.log("Valore: " + n);
}

array.forEach((n) => {
    stampa(n);
});
```

Manipolare gli array (1)

La procedura vista in precedenza, applica la data funzione f a tutti i valori dell'array ma senza modificare i valori dell'array originale.

Se invece si vuole che la funzione f modifichi il valore all'interno dell'array:

```
1  let array = [1, 2, 3, 4, 5];
2  array = array.map((n) => n + 1);
3
4  // array ==> [2, 3, 4, 5, 6]
```

La funzione `.map()` prende come parametro una funzione f che verrà eseguita a tutti gli elementi dell'array.

Manipolare gli array (2)

La scrittura `(n => n + 1)` è un modo per definire in modo rapido una funzione.

In maniera estesa:

```
1  let f = ((n) => n + 1);  // typeof [...] = 'function'
2  console.log(f(1));       // 2
```

Nel caso specifico:

- `(n)` indica il parametro (possono essere anche più di uno);
- `n + 1` è il corpo della funzione.

Matematicamente:

$$f(n) = n + 1$$

Manipolare gli array (3)

Questa scrittura può tornare utile ad esempio quando si vuole rappresentare velocemente un'espressione matematica.

Ad esempio:

$$f : \mathbb{N} \to \mathbb{N}, f(x) = x^2 + 3x + 4$$

Diventa:

```
1 let f = (x => x**2 + 3 * x + 4);
```

Manipolare gli array (4)

Un'altra funzionalità degli array è la capacità di filtrare un sottoinsieme dell'insieme.

Osserviamo il problema:

"Sia dato un insieme $A = \{1, 2, 3, 4, 5\}$. Si vuole ricavare un sottoinsieme $A' \subset A$ tale che A' contenga solo gli elementi pari di A."

Matematicamente, si può definire A' in questo modo:

$$A \supset A' = \{a \in A \mid n \bmod 2 \equiv 0\}$$

Manipolare gli array (5)

La funzione f (la condizione $n \bmod 2 \equiv 0$) possiamo "tradurla" in JavaScript nel seguente modo:

```
1  let f = (n => n % 2 == 0);
```

Tornando al problema originale, vi è la necessità di verificare se tutti gli elementi $n \in N$ soddisfino o meno la condizione di f, ma questa volta eliminando gli elementi che invece non la soddisfano.

Manipolare gli array (6)

In JavaScript, possiamo usare la funzione `array.filter([...])` per filtrare i valori di un array basandosi su una data condizione.

Implementando il problema originale:

```
let array = [1, 2, 3, 4, 5];
let f = (n => n % 2 == 0);

array = array.filter(f);
console.log(array);

// [2, 4]
```

Gli oggetti in JS (1)

In JavaScript un oggetto rappresenta una struttura dati non indicizzata ma associativa, in cui ogni elemento è memorizzato come coppia chiave–valore.

La forma più comune di definizione è la seguente:

```
let studente = {
    nome: "Leonardo Essam",
    eta: 21,
    corso: "Informatica"
};
```

Gli oggetti in JS (2)

Le proprietà possono essere lette e modificate sia con la *dot notation* che con la *bracket notation*:

```
1  console.log(studente.nome);       // "Leonardo Essam"
2  console.log(studente["corso"]);   // "Informatica"
3
4  studente.eta = 19;                // modifica proprieta'
5  studente["corso"] = "Lettere";    // modifica con bracket
```

La bracket notation è indispensabile quando la chiave non è un identificatore valido o è calcolata dinamicamente.

Gli oggetti in JS (3)

Gli oggetti sono dinamici: possiamo aggiungere o eliminare proprietà in qualsiasi momento.

Ad esempio:

```
1  studente.matricola = "12345";        // aggiunta
2  delete studente.corso;               // rimozione
```

Gli oggetti in JS (4)

Il tipo di dato dei valori di un oggetto non ha limiti: può avere valori, oltre ai tipi di dato primitivi, anche funzioni, altri oggetti, array, ecc.

Ad esempio:

```
let rettangolo = {
    base: 5,
    altezza: 10,
    area: function() {
        return this.base * this.altezza;
    }
};

console.log(rettangolo.area());     // 5 * 10 = 50
```

L'uso di `this` permette di riferirsi alle altre proprietà dello stesso oggetto.

Gli oggetti in JS (5)

Per enumerare le proprietà di un oggetto si possono usare:

- Loop `for ... in`;
- Funzioni come `Object.keys()`, `Object.values()` e `Object.entries()`.

Gli oggetti in JS (6)

Per iterare le coppie di chiave-valore (*key-value*) di un oggetto vale la seguente scrittura:

```
for (let chiave in studente) {
  console.log(chiave + ": "+ studente[chiave]);
}

// nome: Leonardo Essam
// eta: 21
// corso: Informatica
```

Gli oggetti in JS (7)

Per estrarre solo le **chiavi** di un oggetto, vale:

```
let keys = Object.keys(studente);
console.log(keys);

// ['nome', 'eta', 'corso']
```

Per estrarre solo i **valori** di un oggetto, vale:

```
let values = Object.values(studente);
console.log(values);

// ['Leonardo Essam', 21, 'Informatica']
```

Gli oggetti in JS (8)

Per accedere iterativamente alle coppie chiave-valore di un oggetto, si può usare la funzione `Object.entries()`.

Tenendo come riferimento l'oggetto `rettangolo` (slide 18):

```
let entries = Object.entries(rettangolo);

entries.forEach((entry) => {
    console.log(entry);
});

// ['base', 5]
// ['altezza', 10]
// ['area', f]
```

Osserviamo che `entries` è un array di array!

Gli oggetti in JS (9)

Un esempio più complesso:

```
let corso = {
    nome: 'Web Development',
    studenti: [
        {
            nome: 'Anna',
            voto: 28
        },
        {
            nome: 'Luca',
            voto: 30
        }
    ]
};
```

Buonarroti