Javascript – Array

Javascript – array

In Javascript gli array sono collezioni che contengono elementi anche di tipo differente.

Ogni elemento ha una posizione all'interno dell'array e la numerazione parte da 0.

In Javascript gli array sono dinamici, ovvero non hanno una dimensione predeterminata. Aumentano o diminuiscono in base al numero di elementi presenti.

Ci sono molteplici modi per creare un array:

```
var array = [elem1, elem2, … , elemN];

var array = [ ];

var array = new Array();

var array = new Array(10);

var array = new Array(elem1, elem2, … , elemN);
```

Javascript – array

L'accesso ad un elemento dell'array, sia in lettura che in scrittura, avviene specificando l'indice dell'elemento tra parentesi quadre:

```
var array = [elem1, elem2, … , elemN];

array[2] = '12';

var elem1 = array[0];
```

L'aggiunta di un nuovo elemento può avvenire valorizzando un indice precedentemente non esistente, oppure utilizzando uno dei metodi messi a disposizione (vedi prossima slide).

```
var array = [1, 2];

array[2] = 3; //ora l'array ha 3 elementi

array[4] = 5; //ora l'array ha 5 elementi. Il quarto vale undefined
```

Javascript – array

Anche gli array sono oggetti e come tali hanno proprietà e metodi. Tra le proprietà troviamo:

- `length` – restituisce il numero di elementi presenti nell'array

Tra i metodi troviamo invece:

- `concat()` – concatena due o più array restituendone uno nuovo
- `push()` – accoda uno o più elementi ad un array esistente
- `unshift()` – inserisce uno o più elementi all'inizio di un array esistente
- `pop()` – rimuove l'ultimo elemento dell'array e lo restituisce
- `shift()` – rimuove il primo elemento dell'array e lo restituisce
- `splice()` – rimuove elementi dal centro dell'array con la possibilità anche di inserirne

Javascript – array

Tra i metodi troviamo invece:

- `toString()` – Converte l'array in stringa separando gli elementi con la virgola
- `join()` – Converte un array in stringa specificando un separatore
- `slice()` – Estrae una porzione di un array creandone uno nuovo
- `indexOf()` – Restituisce l'indice della prima occorrenza dell'elemento passato come parametro. -1 se non esiste.
- `lastIndexOf()` – Restituisce l'indice dell'ultima occorrenza dell'elemento passato come parametro. -1 se non esiste.
- `includes()` – restituisce true se l'elemento passato come parametro esiste nell'array

NB: Esistono anche altri metodi che prevedono il passaggio di funzioni. Li vedremo in seguito

Javascript – array – Spread Operator

La concatenazione di array può essere scritta più agevolmente usando l'operatore spread, composto da tre punti prima del nome di un array.

L'operatore ha l'effetto di sostituire al nome dell'array tutti i suoi elementi.

```
const array1 = [1,2,3];
const array2 = [4,5,6];

const array3 = [...array1, ...array2];
```

E' equivalente a:

```
const array3 = array1.concat(array2);
```

Javascript – for of

L'operatore for of permette di eseguire del codice su tutti gli elementi di oggetti che rappresentano collezioni (per esempio Array).

```
for(let variabile of object) { … }
```

Per esempio:

```
var array=[1,2,3,4];

for(let elemento of array) { console.log(elemento); }
```

Javascript – Funzioni

Javascript – funzioni

Le funzioni sono blocchi di codice, ai quali è associato un nome, richiamabili da altri punti del codice.

Le funzioni possono avere parametri in ingresso e possono avere valori di ritorno. In javascript si definiscono con la parola chiave function:

```
function nomeFunzione( param1, param2,…, paramN)
{
      istruzioni;
      return retVal;
}
```

Javascript – funzioni

I parametri in ingresso sono da considerarsi come variabili locali alla funzione stessa. Sono cioè utilizzabili all'interno della funzione e all'interno di eventuali funzioni contenute.

Il valore che la funzione restituisce al suo chiamante viene identificato con la parola chiave return.

Return ha anche l'effetto di terminare immediatamente l'esecuzione della funzione

La chiamata di una funzione avviene specificandone il nome e gli eventuali parametri.

```
function somma( x, y)
{
      return x+y;
}
var s = somma(10,7);
```

Javascript – funzioni - parametri

In linea di principio il passaggio dei parametri alle funzioni viene fatto per "valore". Il valore viene cioè copiato all'interno della variabile locale della funzione e le modifiche che questa può subire non si ripercuotono sul chiamante. Esempio:

```
function doppio( x)
{
      x = x*2;
      return x;
}
var n = 10;
var r = doppio(n);
//Quanto vale r? e quanto n?
```

Javascript – funzioni - parametri

Javascript non effettua particolari controlli sui parametri passati ad una funzione.

- Se vengono passati meno parametri di quelli previsti, i mancanti avranno il valore undefined
- Se vengono passati più parametri di quelli previsti, quelli in eccesso vengono ignorati.
- Se vengono passati valori di un tipo non previsto non si genera nessun errore. Eventualmente l'errore nascerà in fase di utilizzo del parametro

Javascript – funzioni - parametri

E' possibile definire dei valori di default per alcuni parametri. In questo modo se il parametro non viene passato assumerà il valore di default anziché undefined.

Si definisce un valore di default semplicemente specificandolo nella firma della funzione:

```
function nomeFunzione( param1, param2 = 10)
```

Se non specificato il param2 varrà 10. I parametri con valore di default devono necessariamente essere gli ultimi. Un parametro senza valore di default non può seguire uno che ce l'ha.

La sintassi è equivalente a scrivere:

```
function nomeFunzione( param1, param2) {
        param2 = param2 || 10;
}
```

Javascript – funzioni come dati elementari

In Javascript le funzioni sono considerate come dati elementari. In particolare è possibile:

- Assegnare una funzione ad una variabile. La funzione sarà poi invocabile tramite il nome della variabile
- Passare una funzione come parametro di un'altra funzione. La funzione che riceve il parametro può invocare la funzione che gli viene passata.
- Creare funzioni anonime, ovvero senza nome, utilizzabili senza doverle salvare.

Javascript – funzioni come dati elementari

- Assegnare una funzione ad una variabile. La funzione sarà poi invocabile tramite il nome della variabile. La chiamata può avvenire solo dopo l'assegnazione, come previsto per tutte le variabili.

```
function somma( x, y) { return x+y; }


var s = somma;
console.log( s(3,5) );


var quadrato = function(x) { return x*x };
console.log( quadrato(10) );
```

Javascript – funzioni come dati elementari

- Passare una funzione come parametro di un'altra funzione. La funzione che riceve il parametro può invocare la funzione che gli viene passata

```
function somma( x, y, logger ) {
      var s = x+y;
      logger(s);
      return s;
}
function logToConsole(n) { console.log(n); }
var s = somma(10,20, logToConsole);
```

Javascript – funzioni come dati elementari

- Creare funzioni anonime, ovvero senza nome, utilizzabili senza doverle salvare.

```
function somma( x, y, logger ) {
      var s = x+y;
      logger(s);
      return s;
}
var logToConsole = function(n) { console.log(n); }
var s = somma(10,20, logToConsole);
```

Javascript – funzioni come dati elementari

- Creare funzioni anonime, ovvero senza nome, utilizzabili senza doverle salvare.

```
var s = somma(10,20, function(n) { console.log(n); } );
```

E' possibile specificare funzioni anonime con una forma più compatta, detta lambda expression o arrow functions: La funzione anonima `function(n) { console.log(n); }` può essere riscritta come: `(n) => { console.log(n); }`

In generale una funzione anonima `function(p1,p2…,pn) { istruzioni }` può essere riscritta come:

```
(p1,p2…,pn) => { istruzioni }
```

Javascript – funzioni come dati elementari

In generale una funzione anonima `function(p1,p2…,pn) { istruzioni }` può essere riscritta come: `(p1,p2…,pn) => { istruzioni }`

Se c'è solo un parametro è possibile omettere le parentesi tonde: `p1 => { istruzioni }`

Se c'è solo una istruzione è possibile omettere le parentesi graffe: `p1 => istruzione`

Se c'è solo una istruzione il suo risultato sarà considerato come valore di ritorno della funzione anonima. Non serve specificare la parola chiave return

Javascript – array - Find

Alcuni metodi degli array prevedono il passaggio di una funzione. Per esempio:

- find( ) – restituisce il primo elemento dell'array che soddisfa determinate condizioni

Il metodo find prevede che gli venga passato una funzione che:

- Riceve in input un singolo elemento dell'array
- Restituisce un booleano che vale true se l'elemento soddisfa la ricerca, altrimenti false

La funzione passata come parametro verrà eseguita per ogni elemento. Verrà restituito il primo elemento dell'array per il quale la funzione ha restituito true.

Javascript – array - Find

La funzione può essere passata in più modi:

1. Passando il nome di una funzione esistente

```
function criterioDiRicerca(element)
{
        return element >0;
}

const array = [-1,0,1,2];
const risultato = array.find(criterioDiRicerca);
```

Javascript – array - Find

La funzione può essere passata in più modi:

2. Passando una funzione anonima creata al volo

```
const array = [-1,0,1,2];
const risultato = array.find(function(element)
{
        return element >0;
});
```

3. Passando una arrow function

```
const array = [-1,0,1,2];

const risultato = array.find( element => element > 0 );

const risultato = array.find( e => e > 0 );
```

Javascript – array - Sort

Alcuni metodi degli array prevedono il passaggio di una funzione. Per esempio:

- sort( ) – ordina l'array usando, come criterio di ordinamento, la funzione passata come parametro. Se non viene passato nessun parametro l'ordinamento è alfabetico. L'ordinamento avviene direttamente sull'array e viene anche restituito un riferimento all'array stesso

Il metodo sort prevede che gli venga passato una funzione che:

- Riceve in input due elementi dell'array
- Restituisce un valore numerico < 0 se il primo elemento è minore del secondo, zero se sono uguali, > 0 se il primo elemento è > del secondo

La funzione passata come parametro verrà eseguita per ogni coppia di elementi, fino a determinare il nuovo ordinamento dell'array

Javascript – array - Sort

La funzione può essere passata in più modi:

1. Passando il nome di una funzione esistente

```
function criterioDiOrdinamento(element1, element2)
{
        return element2 – element1;
}

const array = [-1,0,1,2];
array.sort(criterioDiOrdinamento);
```

NB: Quale ordinamento stiamo generando con questo codice?

Javascript – array - Sort

La funzione può essere passata in più modi:

2. Passando una funzione anonima creata al volo

```
const array = [-1,0,1,2];
array.sort(function(element1,element2)
{
        return element2 – element1;
});
```

3. Passando una arrow function

```
const array = [-1,0,1,2];

array.sort( (element1,element2) => element2 – element1);
```

Javascript – array - forEach

Alcuni metodi degli array prevedono il passaggio di una funzione. Per esempio:

- forEach( ) – esegue la funziona passata come parametro per ogni elemento dell'array, procedendo per ordine, dal primo all'ultimo elemento..

Il metodo forEach prevede che gli venga passato una funzione che:

- Riceve in input un elemento dell'array
- Definisce il codice da eseguire su quell'elemento

Javascript – array - forEach

La funzione può essere passata in più modi:

1. Passando il nome di una funzione esistente

```
function criterioEsecuzione(element)
{
        console.dir(element);
}

const array = [-1,0,1,2];
array.forEach(criterioEsecuzione);
```

Javascript – array - forEach

La funzione può essere passata in più modi:

2. Passando una funzione anonima creata al volo

```
const array = [-1,0,1,2];
array.forEach(function(element)
{
        console.dir(element);
});
```

3. Passando una arrow function

```
const array = [-1,0,1,2];

array.forEach( element =>        console.dir(element));
```

Javascript – array - every

Alcuni metodi degli array prevedono il passaggio di una funzione. Per esempio:

- every( ) – Determina se tutti gli elementi dell'array soddisfano la funziona passata come parametro.

Il metodo every prevede che gli venga passato una funzione che:

- Riceve in input un elemento dell'array
- Restituisce true se l'elemento soddisfa il criterio

Il metodo every restituisce true solo se la funzione ha restituito true per ogni elemento. Nel momento in cui un elemento restituisce false, quelli successivi non vengono più valutati

Javascript – array - every

La funzione può essere passata in più modi:

1. Passando il nome di una funzione esistente

```
function criterioTuttiPari(element)
{
        return element % 2 == 0
}

const array = [-1,0,1,2];
const risultato = array.every(criterioTuttiPari)
```

Javascript – array - every

La funzione può essere passata in più modi:

2. Passando una funzione anonima creata al volo

```
const array = [-1,0,1,2];
const risultato = array.every(function(element)
{
        return element % 2 == 0;
});
```

3. Passando una arrow function

```
const array = [-1,0,1,2];

const risultato = array.every( element => element % 2 == 0);
```

Javascript – array

Alcuni metodi degli array prevedono il passaggio di una funzione. Per esempio:

- some( ) – Determina se almeno un elemento dell'array soddisfa la funziona passata come parametro. La funzione deve avere la stessa firma prevista per every
- filter( ) – Restituisce un array con tutti gli elementi che soddisfano la funzione passata come parametro. La funzione deve avere la stessa firma prevista per find
- map ( ) – restituisce un array in cui ogni elemento è il risultato dell'elaborazione effettuato dalla funzione passata come parametro. La funzione riceve come input un elemento dell'array (più alcuni parametri opzionali) e restituisce l'elemento opportunamente elaborato
- reduce( ) – restituisce una "somma" calcolata su tutti gli elementi di un array. La somma viene calcolata con una funzione passata come parametro, con firma composta da due parametri, un totalizzatore e il singolo elemento dell'array. La funzione deve restituire il nuovo valore per il totale. Reduce accetta come secondo parametro opzionale il valore iniziale per il totalizzatore