Tipi di dato strutturati: Array

- I dati visti finora sono:
  - numeri (interi o razionali),
  - booleani
  - le stringhe (sequenze di caratteri)
- ma i dati manipolati nelle applicazioni reali sono spesso complessi (o strutturati)
- Gli array permettono di definire dati strutturati
  - sono composti da elementi, (che nei linguaggi con i tipi sono omogenei), in JavaScript non è detto.
  - ogni elemento è identificato all'interno dell'array da un numero d'ordine detto indice dell'elemento
  - il numero di elementi dell'array è detto lunghezza (o dimensione) dell'array
  - Gli array di JavaScript sono dinamici, ovvero il numero dei loro elementi è variabile.
- Consentono di rappresentare tabelle, matrici, matrici n-dimensionali.

Un esempio

- Supponiamo di dover rappresentare e manipolare la classifica di un campionato cui partecipano 16 squadre.
- È del tutto naturale pensare ad una tabella

Classifica

Squadra A	Squadra B	...	Squadra C
1^o posto	2^o posto		16^o posto

che evolve con il procedere del campionato

Classifica

Squadra B	Squadra A	...	Squadra C
1^o posto	2^o posto		16^o posto

Dichiarazione di variabile e inizializzazione ad un array

Sintassi: dichiarazione di variabile array, ci sono varie alternative:

- dichiarazione di un array vuoto:

  ```
  var nomeArray = [];
  var nomeArray = new Array();
  ```
  oppure equivalente:

  Esempio: `var vet=new Array();` equivalente a `var vet=[];`

- dichiarazione della variabile e inizializzazione degli elementi con il valore `undefined`.

  ```
  var nomeArray = new Array(k);
  ```

  Esempio: `var vet=new Array(8);`

  la variabile `vet` è dichiarata e inizializzata ad un array di 8 elementi `undefined`.

Dichiarazione di variabile e inizializzazione ad un array

- dichiarazione di variabile array e inizializzazione degli elementi:

  var `nomeArray = [espressione`$_1$`,...,espressione`$_k$`];`

  Esempio: **var** `vet=[2,4,5,-6,-3,0];`

- oppure, equivalente alla precedente:

  var `nomeArray =` **new** `Array(espressione`$_1$`,..,espressione`$_k$`;)`

  Esempio: **var** `vet=`**new** `Array(2,4,5,-6,-3,0);`

  In entrambi i casi viene dichiarata la variabile array ed inizializzata con i 6 valori interi (2,4,5,-6,-3,0).

- Una rappresentazione grafica degli array è la seguente:

  Esempio: Se ho dichiarato **var** `vet= new Array(6);`:

indice	elemento	variabile
0	`undefined`	`vet[0]`
1	`undefined`	`vet[1]`
2	`undefined`	`vet[2]`
3	`undefined`	`vet[3]`
4	`undefined`	`vet[4]`
5	`undefined`	`vet[5]`

- Ogni elemento del vettore è denotabile usando il nome del vettore e un indice risultante dalla valutazione di un `espressione`

  Sintassi: elemento di array

  ```
  nomeArray[espressione];
  ```

- Gli array sono indicizzati a partire da 0. Quindi gli indici di un array di 6 elementi varieranno nell'intervallo (*range*) $[0, 5]$
- `vet[i]` è l'elemento del vettore `vet` di indice `i`.

Modifica di un array

- La modifica degli elementi di un vettore utilizza il comando di assegnamento, ponendo a sinistra l'espressione che denota l'elemento di un array. Ad esempio:

```
var index = 2;
vet[index+1])=asknum();
```

  Assegna al 4^o elemento (che ha indice 3) il valore letto dall'input.

- **Attenzione**: l'espressione che compare all'interno delle parentesi ([]) può essere una qualunque espressione purchè calcoli un intero (≥ 0).
  Esempio:

```
var index = 2;
vet[index+1]=0;
```

  se `foo` è una funzione senza argomenti che calcola un intero ≥ 0 anche:

```
var x=vet[foo()];
```

  è corretto.

Esempi di istruzioni che coinvolgono array

Supponendo ad esempio di avere dichiarato:

```
var vet =new Array(6);
```

avremo:

indice	elemento	variabile
0	undefined	vet[0]
1	undefined	vet[1]
2	undefined	vet[2]
3	undefined	vet[3]
4	undefined	vet[4]
5	undefined	vet[5]

Proviamo ad eseguire le seguenti istruzioni e vediamo come cambia il contenuto dell'array.

```
vet[0] = 15;
vet[1] = vet[0] + 4;
print(vet[0] + vet[1]);
```

Esempi di istruzioni che coinvolgono array

- dopo l'esecuzione di `vet[0] = 15;` avremo:

indice	elemento	variabile
0	15	`vet[0]`
1	`undefined`	`vet[1]`
2	`undefined`	`vet[2]`
3	`undefined`	`vet[3]`
4	`undefined`	`vet[4]`
5	`undefined`	`vet[5]`

- dopo l'esecuzione di `vet[1] = vet[0] + 4;` avremo:

indice	elemento	variabile
0	15	`vet[0]`
1	19	`vet[1]`
2	`undefined`	`vet[2]`
3	`undefined`	`vet[3]`
4	`undefined`	`vet[4]`
5	`undefined`	`vet[5]`

Lo stato con gli array

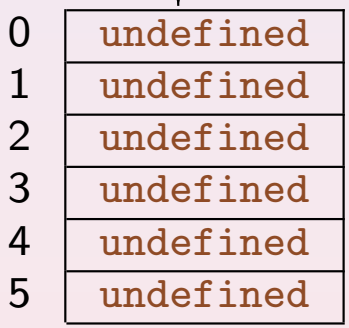
- Lo stato viene esteso: è necessario avere:
  - L'ambiente che rappresenta le variabili come insieme di coppie.
  - Lo heap in cui sono memorizzati gli array (e poi anche gli *oggetti* che vedrete più avanti)
- Le variabili che assumono come valore array (o oggetti) assumono nell'ambiente un valore speciale detto reference che individua l'area dello heap in è memorizzato l'array.
- Questa area di memoria sarà costituita da più locazioni (di memoria) per ogni dato strutturato ed è rappresentata per un singolo array come mostrato precedentemente.
- il reference è spesso indicato graficamente con una freccia oppure con ref_i dal momento che l'effettivo indirizzo non ci interessa ed in generale dipende dal *gestore della memoria* e quindi dall'implementazione.

Esempio di stato

```
var vet1= new Array(6);
```

Stato risultante:

{(vet1,)}

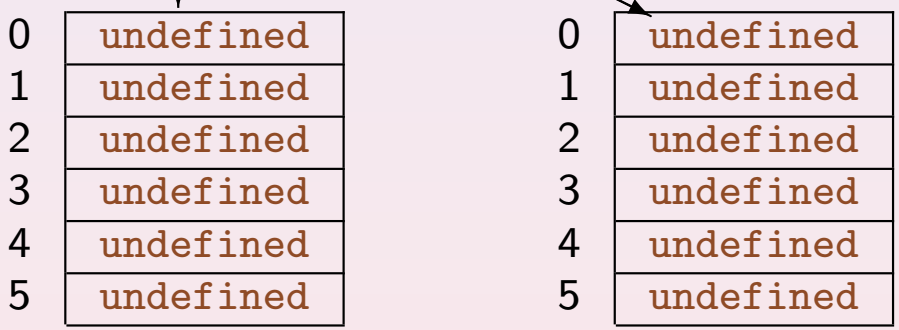


Esempio di stato

```
var vet1= new Array(6);
var vet2= new Array(6);
```

Stato risultante:

{(vet1,), (vet2,)}

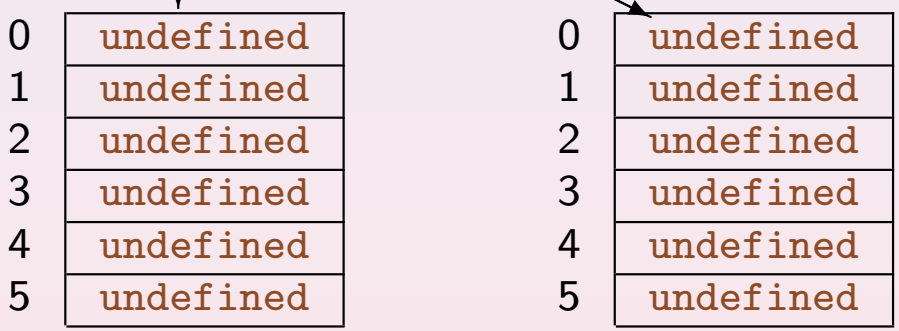


Esempio di stato

```
var vet1= new Array(6);
var vet2= new Array(6);
var a=0;
```

Stato risultante:

{(vet1,), (vet2,), (a,0)}

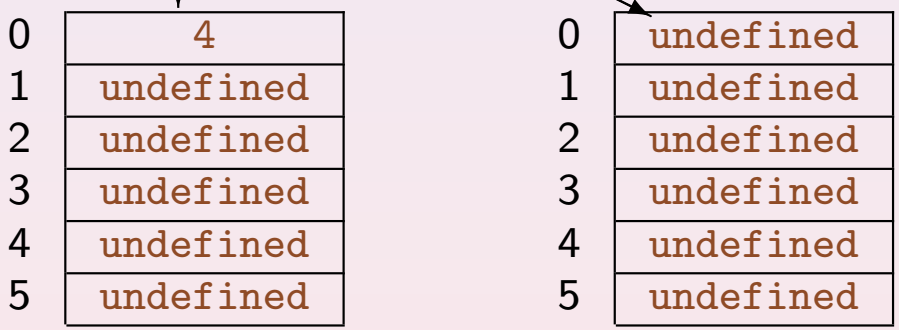


Esempio di stato

```
var vet1= new Array(6);
var vet2= new Array(6);
var a=0;
vet1[0]=4;
```

Stato risultante:

{(vet1,), (vet2,), (a,0)}

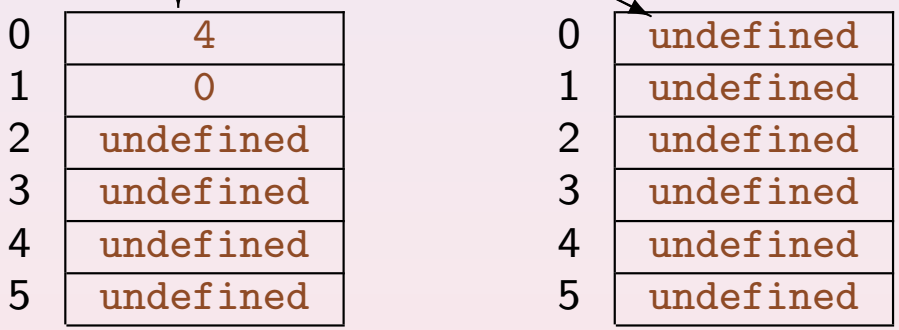


Esempio di stato

```
var vet1= new Array(6);
var vet2= new Array(6);
var a=0;
vet1[0]=4;
vet1[1]=a;
```

Stato risultante:

{(vet1,), (vet2,), (a,0)}

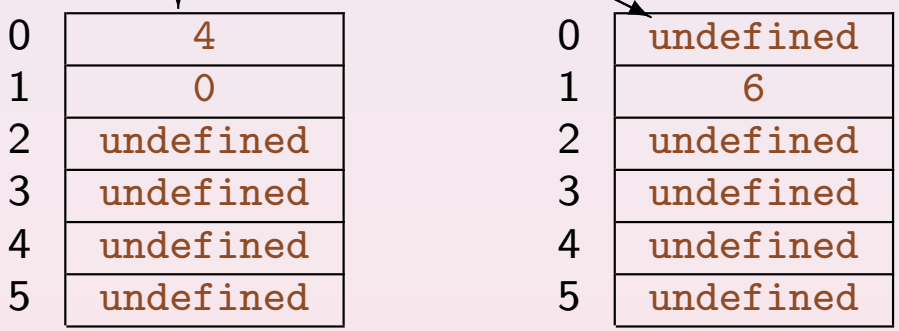


Esempio di stato

```
var vet1= new Array(6);
var vet2= new Array(6);
var a=0;
vet1[0]=4;
vet1[1]=a;
vet2[1]=vet1[1]+2;
```

Stato risultante:

{(vet1,), (vet2,), (a,0)}

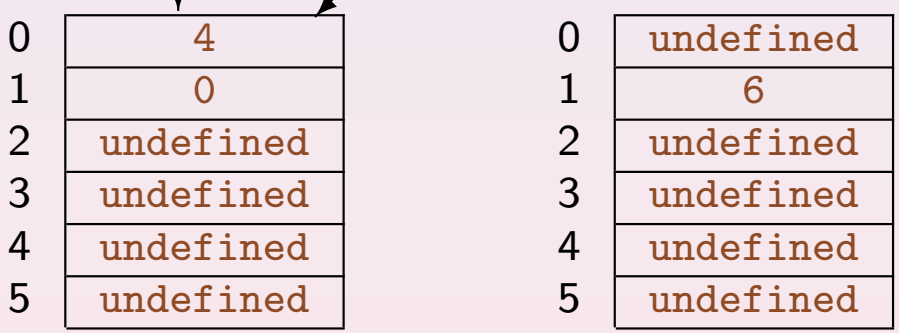


Esempio di stato

```
var vet1= new Array(6);
var vet2= new Array(6);
var a=0;
vet1[0]=4;
vet1[1]=a;
vet2[1]=vet1[1]+2;
vet2=vet1;
```

Stato risultante:

{(vet1,), (vet2,), (a,0)}



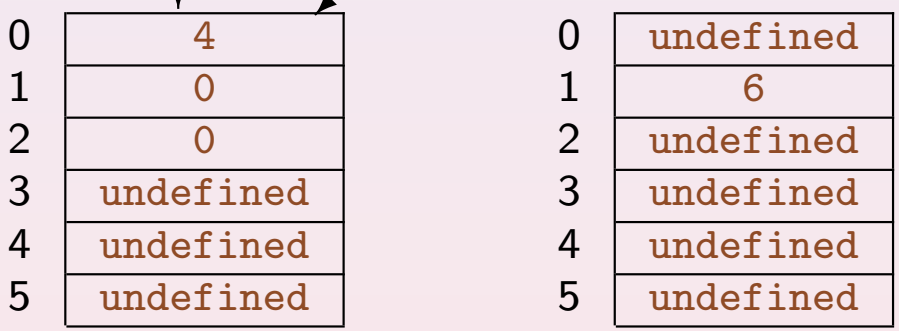
`vet1` e `vet2` condividono lo stesso array cioè hanno lo stesso reference nell'ambiente

Esempio di stato

```
var vet1= new Array(6);
var vet2= new Array(6);
var a=0;
vet1[0]=4;
vet1[1]=a;
vet2[1]=vet1[1]+2;
vet2=vet1;
vet1[2]=vet2[1];
```

Stato risultante:

{(vet1,), (vet2,), (a,0)}



- La modifica di `vet2` modifica anche il valore di `vet1` dal momento che hanno lo stesso reference nell'ambiente!
- l'area di memoria puntata dal vecchio reference di vet2 non è più accessibile.

Osservazioni sugli array

- Anche l'operatore di uguaglianza opera sui reference: due variabili array sono uguali se condividono la stessa struttura.
- **Esempio:** Nell'esempio sopra `vet1==vet2` calcola `true`, nello stato risultante dopo l'assegnamento `vet2=vet1` e `false` prima di tale assegnamento.
- Gli operatori di confronto non sono definiti su array come ragionevole, ma Attenzione confrontare due array non genera errore ma calcola sempre *false*.
- Nell'interprete EasyJS la print funziona anche sugli array:
  Esempio:

```
var LUNG=asknum(), i;
    var vet =new   Array(LUNG);
    ...
    print(vet);
```

  Stampa gli elementi di `vet` separati da virgole.

Manipolazione di vettori

- Avviene molto spesso attraverso cicli e prevalentemente si utilizza l'iteratore **for**.
- L'indice del ciclo varia da 0 a lunghezza-1 per scandire tutto l'array e viene utilizzato per indicare l'elemento dell'array.
- Oggi vediamo dei frammenti di programmi, la prossima lezioni parliamo delle funzioni che hanno come parametri gli array e quindi del passaggio dei parametri nel caso degli array.

  Esempio: Lettura e stampa di un vettore.

```
 var LUNG=asknum(), i;
var vet =new   Array(LUNG);
/* vettore di LUNG elementi, indicizzati da 0 a LUNG-1 */
for (i = 0; i < LUNG; i++) v[i]=asknum();
/* Lettura di un vettore di LUNG elementi.  */
for (i = 0; i < LUNG; i++)
      print("elemento "+i+" "+ v[i]);
/* Stampa di un vettore di LUNG elementi.  */
```

Array dinamici

- In JavaScript gli array sono strutture dinamiche questo vuol dire che il numero degli elementi di un array può cambiare durante l'esecuzione del programma.

  Esempio:

```
 var LUNG=asknum(), i; var vet =new   Array(LUNG);
for (i = 0; i < LUNG; i++) v[i]=asknum();
```

- Per aggiungere un elemento è possibile assegnare un valore ad un elemento con un indice $i >=$LUNG, lunghezza dell'array:
  Esempio: `vet[LUNG]=asknum();`
  Aggiunge un elemento all'array `vet` di indice `LUNG` e valore letto dall'input.

  Esempio: `vet[LUNG+2]=asknum();`
  Aggiunge 2 elementi all'array `vet` uno di indice `LUNG`+1 e valore `undefined` e uno di indice `LUNG`+2 e valore letto dall'input.

Array dinamici

- Dal momento che gli array sono strutture dinamiche, la lunghezza di un array cambia dinamicamente.
- Per questo in JavaScript la dimensione di un array è determinata invocando una funzione predefinita la cui sintassi, deriva dalla sintassi degli oggetti ed è:

  `nomeArray.length`

  dove `nomeArray` è il nome della variabile a cui è associato l'array

  Esempio:

```
 var LUNG=asknum(), i;
var vet =new   Array(LUNG);
for (i = 0; i < vet.length; i++) v[i]=asknum();
```

`vet.length` risulta nell'esempio sopra uguale a `LUNG`, ma se aggiungo elementi a `vet`, `vet.length` cambierà.

Esempi

Inizialmente, consideriamo esempi con array il cui numero di elementi resta costante. In questo caso, molte delle operazioni che abbiamo gia' visto su sequenze di elementi possono essere facilmente modificate per applicarle agli elementi di un array.

- Calcolo della somma degli elementi di un vettore. **var** `a = new Array(10), i, somma = 0;`

```
...
for (i = 0; i < a.length; i++) somma = somma+a[i];
print(somma);
```

- Calcolo del massimo degli elementi di un vettore. **var** `a = new Array(10), i;`

```
    ...
var max=v[0];
for (i = 1; i < a.length; i++)
      if (max<a[i]) max=a[i];
print(max);
```

Ricerca di un elemento in un vettore

Dati un array di `LUNG` elementi e un valore `v` cercare se esiste un elemento uguale a `v` nell'array. Il problema ha vari aspetti:

- Cosa calcola il programma/funzione:
  - può stampare un messaggio, la funzione (quando la definiremo) restituisce true o false
  - in alternativa si può calcolare la posizione $>= 0$, nel caso della funzione, se il valore non viene trovato, calcola un valore negativo (ed es.-1).
  - Un'altra versione prevede il calcolo del numero di occorrenze del valore.
- L'array non è ordinato: la soluzione è molto simile a quella del problema dell'esercizio 4.2 che cerca se esiste il valore 0 in una sequenza di k interi letta in input. Le differenze sono che i valori della sequenza sono già memorizzati nell'array e il valore da cercare non è necessariamente 0 ma un generico valore v.
- L'array è ordinato: la soluzione può essere più efficiente. Le vedremo tra poco.

Una soluzione del problema della ricerca

L'array non è ordinato e il programa stampa un messaggio che indica la posizione se esiste.

```
var a= new Array(5);
var i=0, v=asknum(), trovato=false;
  //...inizializzazione dell'array
while ( i<a.length && !trovato)
  if (a[i]==v) trovato=true;
  else i++;
if (trovato) print(v+"trovato in posizione "+i);
else print(v+"non trovato");
```

Il numero di elementi esaminati è sempre *lineare* con la lunghezza dell'array.

Shift a sinistra degli elementi di un vettore

- Spostamento a sinistra di una posizione di tutti gli elementi.

  ad esempio se `a` è l'array di `LUNG`=5 elementi seguente:

2	4	6	8	10
0	1	2	3	4

è necessario che a[0]=a[1]:

4	4	6	8	10
0	1	2	3	4

a[1]=a[2]:

4	6	6	8	10
0	1	2	3	4

... a[3]=a[4]

4	6	8	10	10
0	1	2	3	4

in generale a[i]=a[i+1] per i[0,LUNG-2]

Shift a sinistra

Dalle osservazioni precedenti risulta evidente che dobbiamo scrivere un ciclo:

```
for (i=0; i<a.length-1; i++) a[i]=a[i+1];
```

Ma...cosa accade del primo e ultimo elemento? Nella soluzione sopra il primo elemento viene perso e ultimo e penultimo elemento risultano uguali.

Alternativamente possiamo definire lo shift *circolare*, in cui il primo elemento viene memorizzato nell'ultima posizione. In questo caso nell'esempio precedente avremo:

4	6	8	10	2
0	1	2	3	4

Per esercizio, completare il programma per lo shift a sinistra e lo shift circolare a sinistra. Scrivere due programmi per lo shift e lo shift circolare a destra.

- Leggere N interi e stampare i valori maggiori di un valore intero y letto in input, dopo la lettura degli N interi.

```
var ris= new   Array (N);
 var y, i;
 for (i = 0; i < N; i++) ris[i]=asknum();
 y=asknum() ;
 print("Valori maggiori di "+ y);
 for (i = 0; i < N; i++)
   if (ris[i] > y)   print(v[i]);
```