Laboratorio I
a.a. 2025/2026

Object & Array

Contenuti

- Oggetti
  - Denotazione
  - Esempi, uso come dizionari
  - Qualche programma con gli oggetti
- Array
  - Denotazione
  - Esempi, uso come liste
  - Qualche programma con gli Array

Tipi di dato e valori letterali

Ricordate questo lucido?

JavaScript consente di denotare **valori letterali** di determinati tipi:

- **Boolean:** true, false
- **Undefined:** undefined (→ non è stato definito nessun valore)
- **Null**: null (→ so che non c'è un valore)
- **Number**: 1, 5, -53.38, 12.3e4 (numeri "normali", $-(10^{53}-1) \le n \le 10^{53}-1$)
- **BigInt**: 900719925474343441698716109 92n (interi a precisione infinita)
- **String**: "che bel castello", "हिन्दू", 'hello', `x vale ${x}` (sequenze di caratteri)
- **Object**: { nome: "Pino", età: 19 } (mappe da chiavi a valori - anche dizionari)
  - **Array**: [1, 5, 8, 12, 21, 33], [1, "ciao!", { x: 12, y:22 }, ["banana"]] (liste con indici numerici)
  - **Function**: x=>2*x, (a,b) => a+b, x=> { if (x>0) return x; else return -x } (funzioni)

Oggetti (o *dizionari*)

Un **oggetto** (o dizionario) è una *mappa* da **chiavi** a **valori**

Qui *mappa* è usato nel senso matematico: in pratica, è una funzione matematica il cui dominio è l'insieme di tutte le chiavi (in pratica: stringhe e numeri), e il codominio è l'insieme di tutti i valori possibili in JavaScript.

Potremmo anche dire che un oggetto è un insieme di coppie (chiave, valore).

Per esempio, un oggetto *pippo* potrebbe essere indicato in stile matematico con

pippo = { ("passione", "Fondamenti"), ("età", 35), ("capelli", true) }

Oggetti (o *dizionari*)

In JavaScript, la *denotazione* per i valori di tipo oggetto è un po’ più comoda:

pippo = **{** passione**:** "Fondamenti"**,** età**:** 35**,** capelli**:** true **}**

Gli oggetti consentono quindi di raccogliere in un solo valore (di tipo oggetto) molti valori diversi (di qualunque tipo), assegnando a ciascuno di essi un nome (la chiave)

Alcuni esempi di letterali di tipo oggetto

{ } – oggetto vuoto

{ nome: "Pippo", età: 35 } – il caso più comune

{ x } – *espansione di variabile:* se *x* vale 5, è come { x: 5 }

{ "nome": "Pippo", "età" : 35 } – le chiavi possono essere stringhe

{ "!!": 5, "😎": true } – anche stringhe che non sono identificatori

{ 0: 6, 1: 4, 2: 12 } – le chiavi possono essere numeri

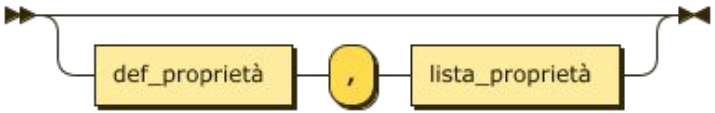
{ ["pip"+fiume]: true } – *chiave calcolata:* se *fiume* vale "po", è come { pippo: true }

Costanti letterali di tipo oggetto - Sintassi

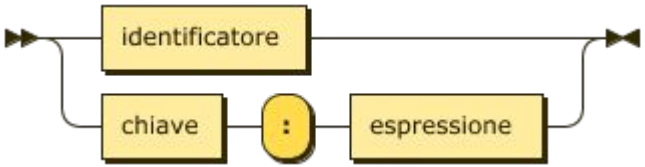
letterale_oggetto ::= '**{**' *lista_proprietà* '**}**'
lista_proprietà ::= | *def_proprietà* '**,**' *lista_proprietà*
def_proprietà ::= identificatore | *chiave* '**:**' *espressione*
chiave ::= identificatore
| *letterale_stringa*
| *letterale_numero*
| '**[**' *espressione* '**]**'

letterale_oggetto:

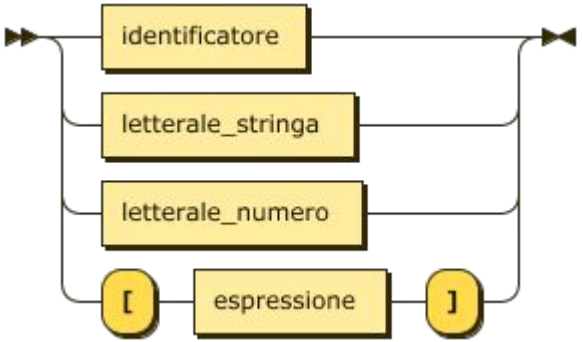
lista_proprietà:



def_proprietà:



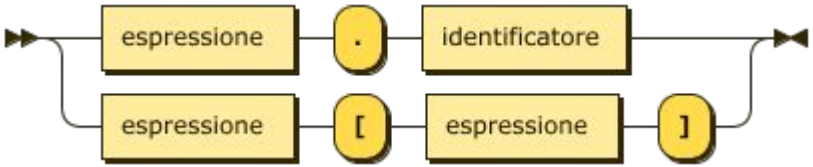
chiave:



Operazioni sugli oggetti

L'operazione principale sugli oggetti è l'**accesso** a un membro (chiave)

espressione_accesso:



espressione_accesso ::= espressione '**.**' *identificatore*
| *espressione* '**[**' *espressione* '**]**'

```
let pippo = { nome: "Filippo", età: 35, capelli: true }

pippo.nome → "Filippo"
pippo["nome"] → "Filippo"
pippo.età → 35

if (pippo.capelli) {
     pippo.età++
     pippo.nome="Guendalino"
}

pippo → { capelli: true, età: 36, nome: "Guendalino" }
```

```
pippo.occhi → undefined

pippo.occhi = 2

pippo → { capelli: true, età:
36, nome: "Guendalino",
occhi: 2 }
```

Operazioni sugli oggetti

Altre operazioni utili:

- Controllo se una chiave esiste: *chiave* **in** *oggetto*
- Cancellazione di una coppia chiave-valore: **delete** *chiave*
- Scorrere le chiavi: **for** (*k* **in** *oggetto*) …

```
let pippo = { nome: "Filippo", età: 35, capelli: true }

"nome" in pippo → true
"occhi" in pippo → false

delete pippo.nome
pippo → { capelli: true, età: 35 }

for (var k in pippo)
     delete pippo[k]

pippo → { }
```

Cancello la chiave che è il **valore** corrente di k, e non la chiave che si chiama k

Vedremo usi creativi di queste operazioni negli esercizi!

Array

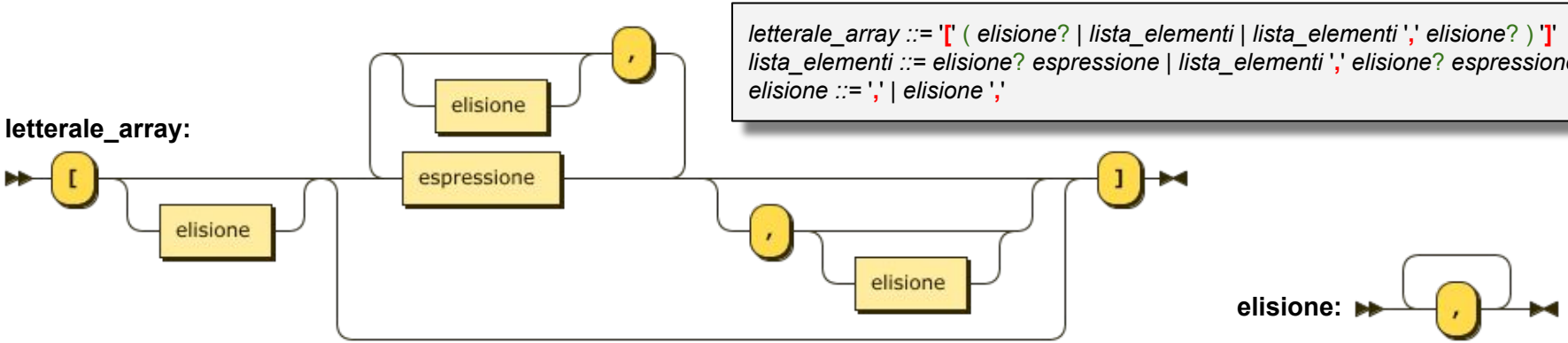
Gli **array** sono un tipo particolare di oggetto, caratterizzato dal fatto che le chiavi sono **numeri interi**

In Informatica, si amano in particolare gli array le cui chiavi sono tutti i numeri interi fra 0 e un numero fissato *n*, e i cui valori sono tutti dello stesso tipo

Nessuna di queste condizioni è però necessaria in JavaScript!

L'ordinamento dei numeri naturali (le chiavi) definisce un ordinamento fra le coppie (chiave,valore): gli array si possono quindi considerare **liste** (ovvero: sequenze ordinate) anziché **insiemi**

Costanti letterali di tipo array - Sintassi



In pratica, è un modo un po' complicato di dire: `[` *espressione* `,` *espressione* `,` … `]`, in cui però una o più delle *espressioni* possono mancare.
Le espressioni mancanti occupano sempre un "posto" nell'array, con il loro indice, ma non hanno un valore. Quelle mancanti in fondo all'array vengono invece ignorate.

Nota: `empty` non fa parte delle espressioni di JS (viene sono mostrato per distinguerlo da *undefined*)

```
let pluto = [ , 10 , , 4 , ]

pluto → [ empty, 10, empty, 4 ]
```

Operazioni sugli array - accesso

chiave **indice**	valore **elemento**
0	“Alina”
1	“Vincenzo”
2	“Giuseppe”
3	“Davide”

Array di lunghezza **4** (indici fra **0** e **3**)

Nel caso degli array, specialmente se “tradizionali”, si usa chiamare le chiavi **indici** e i valori **elementi** dell’array.

Poiché le chiavi sono numeri, e quindi non sono identificatori, l’unico operatore di accesso possibile è quello con le parentesi quadre. In compenso, esiste una notazione più comoda per le costanti letterali di tipo array.

```
var a = [ "Alina", "Vincenzo", "Giuseppe", "Davide" ]

a[0] → "Alina"
a[2]="Peppe"

a → [ "Alina", "Vincenzo", "Peppe", "Davide" ]
a.length → 4
```

Peculiarità degli array in JavaScript

Il fatto che gli array siano in effetti oggetti, consente molta più libertà con la loro definizione.

- Posso avere indici negativi, o che non partono da 0
- Posso avere array sparsi, in cui alcuni elementi mancano
  - Gli assegnamenti con indici non già esistenti creano al volo l'elemento
  - La lettura di valori relativi a indici non esistenti restituisce `undefined`
- Tutti gli array hanno una proprietà implicita **`length`** che restituisce la loro lunghezza corrente (elementi fra 0 e il massimo indice definito)
- **`delete`** cancella un elemento (diventa undefined) ma non cambia la lunghezza
- Assegnare un valore a **`length`** tronca o estende un array

Altre operazioni sugli array

Gli array consentono ovviamente tutte le operazioni consentite sugli object.

In più, hanno una serie di funzioni speciali che semplificano le operazioni su liste.

- **`a.push(z)`** – aggiunge *z* in coda all'array *a*
- **`c=a.concat(b)`** – concatena l'array *b* in coda all'array *a*, restituisce il risultato
- **`z=a.pop()`** – rimuove l'ultimo elemento da *a*, restituisce l'elemento rimosso
- **`x=a.shift()`** – rimuove il primo elemento di *a*, restituisce l'elemento rimosso; tutti gli altri elementi scorrono di un posto per "chiudere il buco"
- **`a.sort()`** – ordina un array in base al valore degli elementi
  - Ma non ditelo a quelli di Programmazione & Algoritmica!

Molte altre operazioni anche più interessanti, ispirate alla **programmazione funzionale**, ma le vedremo negli esempi!

Object e array

Gli array sono un tipo particolare di oggetti, **ma** non sono *esattamente* la stessa cosa

```
var a = [ "Alina", "Vincenzo", "Giuseppe", "Davide" ]
var b = { 0: "Alina", 1: "Vincenzo", 2: "Giuseppe", 3: "Davide" }

a==b → false

for (k in a)
     console.log(a[k]==b[k]) → stampa 4 volte true

for (k in b)
     console.log(a[k]==b[k]) → stampa 4 volte true

typeof a → "object"
typeof b → "object"
```

La differenza sta in effetti nel **tipo**, ma in un senso più profondo di quanto possiamo discutere adesso.

Esercizio 1

(Inventario) Si scriva un programma per gestire un inventario, formato da un insieme di coppie prodotto-quantità. Tali coppie vengono lette da tastiera, fino a quando l'utente inserisce "stop" come nome del prodotto. Stampare la lista dei prodotti inventariati, con le rispettive quantità, su una sola riga.

Esempio

Input: latte, 2, uova, 6, pane, 7, latte, 3, uova, 6, stop

Output: (latte, 5) (uova, 12) (pane, 7)

Esercizio 2

(Media dei voti) Si scriva un programma che, dato un insieme di studenti con i rispettivi voti di 3 esami, calcoli e visualizzi la media aritmetica dei voti di ciascuno. L'utente fornirà il nome di ogni studente, seguito dai suoi 3 voti. L'acquisizione termina quando l'utente inserisce "stop" come nome di uno studente.

Esempio

Input: Luca, 28, 30, 27, Marta, 30, 29, 30, Marco, 18, 22, 25, stop

Output: Luca: 28.33
Marta: 29.67
Marco: 21.67

Q & A