Esercitazione 9 - 23/11

Argomenti:

- **passaggio per valore e per riferimento**: cosa riceve davvero una funzione
- **modificare** un array o **riassegnare** una variabile: due operazioni diverse
- **invertire un array sul posto**, senza array di appoggio
- **stringhe**: indici, `length`, cicli; sono **immutabili**, gli array no; invertire una stringa
- **matrici** (array di array): `m[i][j]` e somma della diagonale

Indice degli esercizi

Esercizio	Argomento
Esercizio 1	Numeri e array passati a una funzione: cosa stampa
Esercizio 2	Inversione di un array senza array di appoggio
Esercizio 3	Stringhe: introduzione
Esercizio 4	Inversione di una stringa e palindromi
Esercizio 5	Matrici: accesso agli elementi
Esercizio 6	Somma delle diagonali di una matrice

Recap 1 · Valore e riferimento

Ogni variabile contiene **una sola cosa**, ma questa cosa può essere di due tipi.

- Con **numeri, booleani e stringhe** la variabile contiene **direttamente il valore**. Con `let y = x;` si **copia il valore**: da quel momento `x` e `y` sono indipendenti.
- Con **array e oggetti** la variabile contiene un **riferimento**, cioè l'«indirizzo» in memoria dell'array. Con `let b = a;` si copia **solo l'indirizzo**: `a` e `b` sono due nomi per lo **stesso** array.

Numeri: la variabile **contiene** il valore, e `let y = x;` ne fa una copia.

```
let x = 5;
let y = x;

   x ┌─────┐        y ┌─────┐
     │  5  │          │  5  │        due scatole, ciascuna col suo 5
     └─────┘          └─────┘
```

Array: la variabile contiene solo l'**indirizzo** dell'array, e `let b = a;` copia l'indirizzo, non l'array.

```
let a = [1, 2];
let b = a;

   a ┌─────┐
     │  ●  ├──────┐
     └─────┘      │
                  ├──►┌─────┬─────┐
                  │   │  1  │  2  │      un solo array
   b ┌─────┐      │   └─────┴─────┘
     │  ●  ├──────┘
     └─────┘
```

Passare un argomento a una funzione equivale a un'assegnazione: il parametro è una nuova variabile che riceve una copia di ciò che c'è nella variabile di chi chiama. Quindi:

- se l'argomento è un **numero**, la funzione riceve una copia del valore: qualunque cosa faccia, l'originale non cambia;
- se l'argomento è un **array**, la funzione riceve una copia del **riferimento**: ha a disposizione lo **stesso array** di chi chiama.

Con un array bisogna quindi distinguere due operazioni molto diverse: **modificare** l'array (`v[0] = 99`) e **riassegnare** la variabile (`v = [99, 99]`). Qui le vediamo per capire cosa succede; **le funzioni che scriverete voi non modificheranno mai gli array che ricevono**: se serve un risultato lo **restituiranno**, costruendo un array nuovo.

```
function modifica(v) {
  v[0] = 99;                                  // MODIFICA l'array: si vede anche fuori
}

function riassegna(v) {
  v = [99, 99];                               // RIASSEGNA la variabile v: ora v indica un
altro array
}                                             // l'array di chi chiama non è toccato

let a = [1, 2, 3];
modifica(a);
console.log(a);                               // [ 99, 2, 3 ]

let b = [1, 2, 3];
riassegna(b);
console.log(b);                               // [ 1, 2, 3 ]
```

In `riassegna` il parametro `v` è una variabile **locale** che per un attimo indicava lo stesso array di `b`; scrivendo `v = ...` la si fa indicare un array nuovo, ma `b` continua a indicare il vecchio.

Il confronto `==` tra due array confronta i **riferimenti**, non il contenuto: è `true` solo se sono **lo stesso array**.

```
let a = [1, 2];
let b = a;
let c = [1, 2];
console.log(a == b);                          // true: stesso array
console.log(a == c);                          // false: due array diversi, anche se con gli
stessi elementi
```

Esercizio 1

Ripasso: Recap 1 · Valore e riferimento.

Dite cosa stampa ciascun programma, **senza eseguirlo**, e spiegate il perché.

A.

```
function incrementa(x) {
  x = x + 1;
  return x;
}

let n = 5;
let m = incrementa(n);
console.log(n, m);
```

B.

```
function incrementaTutti(v) {
  for (let i = 0; i < v.length; i++) {
    v[i] = v[i] + 1;
  }
}

let a = [1, 2, 3];
incrementaTutti(a);
console.log(a);
```

C.

```
function azzera(v) {
  v = [0, 0, 0];
}

let a = [1, 2, 3];
azzera(a);
console.log(a);
```

D. Cosa cambierebbe in C se, al posto di `v = [0, 0, 0];`, la funzione contenesse un ciclo `for` che scrive `0` in ogni posizione di `v`?

E. Scrivere una funzione `zeri(v)` che **restituisce un nuovo array** di zeri della stessa lunghezza di `v`, **senza modificare** `v`. Poi dite cosa stampa

```
let a = [1, 2, 3];
let z = zeri(a);
console.log(a, z);
```

Esercizio 2

Ripasso: Recap 1 · Valore e riferimento (un array è modificabile).

Nell'Esercitazione 8 abbiamo costruito un **nuovo array** con gli elementi di `v` in ordine inverso. Qui invece vogliamo **invertire l'array stesso**, senza usare nessun array di appoggio: niente `[]`, niente copie.

Dato un array `v`, scrivere un **programma** che **modifica** `v` in modo che i suoi elementi siano in ordine inverso, e poi lo stampa. Ad esempio con `v = [1, 2, 3, 4, 5]` si stampa `[ 5, 4, 3, 2, 1 ]`.

Recap 2 · Le stringhe

Una **stringa** è una sequenza di caratteri, scritta tra apici (`"ciao"` o `'ciao'`). Come per gli array, i caratteri hanno un **indice** che parte da `0` :

```
let s = "ciao";
console.log(s.length);                          // 4
console.log(s[0]);                              // c: il primo carattere
console.log(s[s.length - 1]);                   // o: l'ultimo carattere
console.log(s[4]);                              // undefined: oltre la fine
```

Ogni carattere è a sua volta una stringa di lunghezza 1. Per scorrere una stringa si usa un ciclo `for` sugli indici, come per un array:

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

Le stringhe si **concatenano** con `+` e si confrontano con `==` (che confronta il **contenuto**). La stringa vuota è `""` .

```
let saluto = "ciao" + " " + "mondo";             // ciao mondo
console.log("ciao" == "ciao");                  // true
```

Due metodi permettono di passare da una stringa a un array e viceversa:

- `s.split(separatore)` **spezza** la stringa in corrispondenza del separatore e restituisce l'**array** dei pezzi;
- `v.join(separatore)` **unisce** gli elementi dell'array in una **stringa**, mettendo il separatore tra uno e l'altro.

```
let frase = "il gatto nero";
let parole = frase.split(" ");                  // [ 'il', 'gatto', 'nero' ]
console.log(parole.length);                     // 3
console.log(parole[1]);                         // gatto
console.log(parole.join("-"));                  // il-gatto-nero
console.log("ciao".split(""));                  // [ 'c', 'i', 'a', 'o' ]: un elemento per
carattere
```

Le stringhe sono immutabili

Ricordiamo che un valore è **mutabile** se si può modificare «sul posto», **immutabile** se non si può: per ottenere un valore diverso se ne deve creare uno nuovo.

- gli **array** (e gli oggetti) sono **mutabili**: `v[0] = 99` cambia l'array;
- **numeri, booleani e stringhe** sono **immutabili**.

Le stringhe somigliano agli array (hanno `length`, si legge `s[0]`), ma **non si possono modificare**:

```
let s = "ciao";
s[0] = "M";                                 // non fa niente (in alcuni ambienti dà
errore)
console.log(s);                             // ciao
```

Anche i metodi delle stringhe **non modificano** `s`: restituiscono un valore nuovo.

```
let lettere = s.split("");
console.log(s, lettere);                    // ciao [ 'c', 'i', 'a', 'o' ]
```

Per «cambiare» una stringa si costruisce quindi **una nuova stringa**, per esempio con la concatenazione:

```
s = s + "!";                                // crea "ciao!" e la assegna a s: la stringa
"ciao" non è stata toccata
```

Visto che sono immutabili, le stringhe passate a una funzione si comportano come i numeri: la funzione non può modificare l'originale, può solo **restituire una stringa nuova**. È la stessa regola che seguiamo per gli array.

Esercizio 3

Ripasso: Recap 2 · Le stringhe.

Sia `s = "Programmazione"`.

A. Quanto valgono `s.length`, `s[0]`, `s[3]`, `s[s.length - 1]` e `s[14]`?

B. Scrivere un programma che stampi la stringa **un carattere per riga**, preceduto dal suo indice: `0 P`, `1 r`, ...

C. Scrivere una funzione `contaVocali(s)` che restituisce quante vocali minuscole (`a`, `e`, `i`, `o`, `u`) ci sono in `s`. Calcolare `contaVocali(s)`.

D. Cosa stampa questo programma, **senza eseguirlo**?

```
let t = "ciao mondo";
t[0] = "M";
console.log(t);

let parole = t.split(" ");
console.log(t, parole);

t = t + "!";
console.log(t);
```

E. Data la stringa `frase = "il gatto nero"`, scrivere un programma che stampi il **numero di parole** e poi la frase con le parole separate da un trattino (`il-gatto-nero`), usando `split` e `join`. Poi la stessa frase **al contrario**, parola per parola (`nero gatto il`), senza usare metodi per invertire.

Esercizio 4

Ripasso: Recap 2 · Le stringhe (l'ultima parte: costruire una nuova stringa).

Visto che le stringhe non si possono modificare, scriviamo funzioni che **restituiscono una stringa nuova**. Senza usare metodi delle stringhe (solo `length`, `s[i]` e `+`).

A. `sostituisci(s, i, c)`: restituisce una stringa uguale a `s` ma con il carattere in posizione `i` sostituito da `c`. Ad esempio `sostituisci("ciao", 0, "M")` restituisce `"Miao"`. Verificare che `s` sia rimasta com'era.

B. `inverti(s)`: restituisce la stringa con i caratteri in ordine inverso. Ad esempio `inverti("ciao")` restituisce `"oaic"`.

C. Usando `inverti`, scrivere `palindroma(s)` che restituisce `true` se `s` si legge uguale da sinistra a destra e da destra a sinistra (per esempio `"anna"`), `false` altrimenti.

Recap 3 · Le matrici

Una **matrice** è un **array di array**: ogni elemento dell'array esterno è una **riga**, che a sua volta è un array.

```
let m = [
  [1, 2, 3],
  [4, 5, 6]
];

console.log(m.length);                          // 2: il numero di righe
console.log(m[0].length);                       // 3: il numero di colonne (lunghezza di una
riga)
console.log(m[1]);                              // [ 4, 5, 6 ]: la riga 1
console.log(m[1][2]);                           // 6: riga 1, colonna 2
```

`m[i][j]` si legge «nella riga `i`, l'elemento in colonna `j`»: prima si sceglie la riga, `m[i]`, che è un array, poi l'elemento di quell'array. Gli indici partono da `0` sia per le righe sia per le colonne. Una matrice è un array, quindi è **mutabile**: `m[0][1] = 20;` modifica l'elemento.

Per scorrerla servono **due cicli annidati**: quello esterno sulle righe, quello interno sulle colonne.

```
for (let i = 0; i < m.length; i++) {
  for (let j = 0; j < m[i].length; j++) {
    console.log(i, j, m[i][j]);
  }
}
```

Una matrice si dice **quadrata** se ha tante righe quante colonne (`n × n`).

Esercizio 5

Ripasso: Recap 3 · Le matrici.

Data la matrice

```
let m = [
  [2, 7, 1],
  [9, 4, 6]
];
```

A. Quanto valgono `m.length`, `m[0].length`, `m[1]`, `m[1][0]` e `m[0][2]`? Cosa succede con `m[2][0]`?

B. Scrivere un programma che stampi tutti gli elementi nel formato `riga 0, colonna 0: 2`, `riga 0, colonna 1: 7`, …

C. Scrivere un programma che **modifichi** `m` sostituendo ogni elemento con il suo doppio, e poi stampi `m`.

Esercizio 6

Ripasso: Recap 3 · Le matrici (matrice quadrata).

Sia `m` una matrice **quadrata** `n × n`, ad esempio

```
let m = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
```

La **diagonale principale** è formata dagli elementi `m[0][0]`, `m[1][1]`, ..., `m[n−1][n−1]` (nell'esempio `1, 5, 9`); la **diagonale secondaria** da quelli che vanno dall'angolo in alto a destra a quello in basso a sinistra (`3, 5, 7`).

A. Scrivere una funzione `sommaDiagonale(m)` che restituisce la somma degli elementi della diagonale principale.

B. Scrivere una funzione `sommaAntidiagonale(m)` che restituisce la somma degli elementi della diagonale secondaria.

C. Scrivere una funzione `sommaDiagonali(m)` che restituisce la somma degli elementi che stanno **su almeno una** delle due diagonali. Attenzione: se `n` è dispari l'elemento centrale appartiene a entrambe, e va contato **una volta sola**.

Ludovica Pannitto - Università di Pisa - CC BY-NC-SA 4.0