Esercitazione 7 - 09/11

Argomenti:

- **array**: creare un array, indici, `length`
- scorrere un array con un ciclo `for` e `while`
- array e funzioni: array come parametro e come valore restituito
- il passaggio per riferimento: perché le nostre funzioni **restituiscono sempre un nuovo array** invece di modificare quello ricevuto

---

Indice degli esercizi

Esercizio	Argomento
Esercizio 1	Indici e `length`: cosa stampa
Esercizio 2	Scorrere un array: cosa stampa
Esercizio 3	Somme e conteggi con un ciclo `for`
Esercizio 4	Cercare lo zero: prima e ultima posizione
Esercizio 5	Array: trovare gli errori
Esercizio 6	Media e conteggio degli elementi
Esercizio 7	Posizione del massimo
Esercizio 8	Le posizioni di un elemento
Esercizio 9	Costruire un nuovo array: i quadrati
Esercizio 10	Filtrare un array
Esercizio 11	Togliere i duplicati
Esercizio 12	Due array: somma elemento per elemento
Esercizio 13	Due array: confronto delle somme
Esercizio 14	Unione di due insiemi

Strumento aggiuntivo · Python Tutor

Per **vedere passo passo** cosa fa un programma (il valore delle variabili a ogni giro di un ciclo) si può usare **Python Tutor**, che supporta anche JavaScript: <https://pythontutor.com/javascript.html>

Incolla il codice, premi **Visualize Execution** e avanza con **Next**. Prova con questo esempio:

```
let i = 1;
let somma = 0;
while (i <= 4) {
  somma += i;
  i++;
}
console.log(somma);                          // 10
```

Guarda come cambiano `i` e `somma` a ogni giro. È utile anche per controllare le tracce degli esercizi.

Recap 1 · Gli array

Finora ogni variabile conteneva **un solo valore** (uno scalare). Un **array** è una oggetto che contiene **più valori in fila**, come una scatola con tanti scomparti numerati.

```
let città = ["Roma", "Firenze", "Pisa", "Lucca"];
let voti = [28, 30, 25];
let pippo = [];                              // array vuoto: nessun elemento
```

Ogni elemento ha un **indice**, cioè la sua posizione. **Si parte da zero**:

indice	0	1	2	3
`città`	`"Roma"`	`"Firenze"`	`"Pisa"`	`"Lucca"`

```
console.log(città[0]);                       // Roma: il primo elemento
console.log(città[2]);                       // Pisa
città[1] = "Siena";                          // cambia il valore dell'elemento in
posizione 1
console.log(città);                          // [ 'Roma', 'Siena', 'Pisa', 'Lucca' ]
```

La **lunghezza** dell'array (il numero di elementi) si ottiene con `length`, **senza parentesi**:

```
console.log(città.length);                   // 4
console.log(città[città.length - 1]);        // Lucca: l'ultimo elemento
console.log(città[4]);                       // undefined: la posizione 4 non esiste
```

Con `length` elementi gli indici validi vanno da `0` a **`length - 1`**. Leggere fuori da questo intervallo **non dà errore**: dà `undefined`.

Assegnare a una posizione oltre la fine, invece, **allarga** l'array: le posizioni intermedie restano vuote e la lunghezza diventa (indice più grande usato) + 1.

```
let pippo = [];
pippo[2] = 5;                                   // oltre la fine: l'array si allarga
console.log(pippo.length);                      // 3: le posizioni 0 e 1 restano vuote
console.log(pippo[0]);                          // undefined: leggendole si ottiene undefined
console.log(pippo);                             // [ <2 empty items>, 5 ]
```

Le posizioni vuote sono dei «buchi»: non contengono il valore `undefined`, semplicemente **non esistono come elementi** (per questo `console.log` scrive `<2 empty items>`). Leggerle però dà `undefined`, come per qualsiasi posizione fuori dall'array.

Per **aggiungere un elemento in fondo** si assegna alla posizione `length`, che è la prima ancora libera:

```
let v = [4, 7];
v[v.length] = 9;                                // v.length vale 2: si assegna alla posizione
2, in fondo
console.log(v);                                 // [ 4, 7, 9 ]
```

Per **scorrere** un array si usa un ciclo `for` con l'indice come contatore:

```
let voti = [28, 30, 25];
for (let i = 0; i < voti.length; i++) {         // i vale 0, 1, 2: si ferma prima di length
  console.log("voto", i, "=", voti[i]);
}
```

Esercizio 1

Dato l'array:

```
let a = [10, 20, 30, 40, 50];
```

dite cosa stampa ciascuna istruzione, **senza eseguire**.

A. `console.log(a[1]);`

B. `console.log(a.length);`

C. `console.log(a[a.length - 1]);`

D. `console.log(a[a.length]);`

E. `console.log(a[1] + a[2]);`

F. `a[0] = a[4]; console.log(a);`

G. `a[7] = 99; console.log(a.length);`

Esercizio 2

Per ogni programma, dite cosa stampa **senza eseguirlo**. Per il programma B aiutatevi con una tabella (una riga per ogni valore di `i`).

A.

```
let v = [3, 8, 1, 6];
for (let i = 0; i < v.length; i++) {
  console.log(i, v[i]);
}
```

B.

```
let v = [2, 4, 6, 8];
for (let i = 1; i < v.length; i++) {
  v[i] = v[i] + v[i - 1];
}
console.log(v);
```

C.

```
let v = [5, 7, 9];
let s = 0;
for (let i = v.length - 1; i >= 0; i--) {
  s = s * 10 + v[i];
}
console.log(s);
```

Esercizio 3

Dato l'array:

```
let v = [4, -2, 7, 0, 9, -5];
```

scrivere un programma, con un ciclo `for`, che **stampi**:

A. la somma di **tutti** gli elementi di `v`.

B. la somma dei soli elementi **positivi**.

C. quanti elementi sono **pari** (ricordate che `0` è pari).

D. la somma degli elementi che si trovano in posizione **dispari** (indici `1`, `3`, `5`).

Esercizio 4

Cercare lo **zero**. Per ciascun punto provate il programma con `[5, 3, 0, 8, 0]`, con `[1, 2, 3]` e con `[0, 4]`.

A. Scrivere un programma che stampi la **posizione del primo zero** presente nell'array `v`, oppure `-1` se non ce ne sono. Con `[5, 3, 0, 8, 0]` deve stampare `2`.

B. Riscrivere il programma con un ciclo `while` che **si ferma appena trova lo zero** (senza continuare a scorrere l'array inutilmente).

C. Modificare il programma A in modo che stampi la posizione dell'**ultimo** zero. Con `[5, 3, 0, 8, 0]` deve stampare `4`.

Recap 2 · Array e funzioni

Un array può essere **parametro** di una funzione, e una funzione può **restituire** un array.

```
function somma(v) {
  let s = 0;
  for (let i = 0; i < v.length; i++) {
    s = s + v[i];
  }
  return s;
}

console.log(somma([1, 2, 3, 4]));              // 10
console.log(somma([]));                        // 0: array vuoto, il ciclo non parte
```

C'è però una differenza importante rispetto ai numeri. Quando si passa un numero a una funzione, la funzione riceve una **copia**: cambiarla dentro non tocca l'originale. Quando si passa un array, la funzione riceve **lo stesso array** (non una copia): se lo modifica, la modifica si vede anche fuori.

```
function raddoppia(v) {
  for (let i = 0; i < v.length; i++) {
    v[i] = v[i] * 2;                          // modifica l'array originale!
  }
  return v;
}

let a = [1, 2, 3];
let b = raddoppia(a);
console.log(a);                               // [ 2, 4, 6 ]: a è cambiato, anche se non
volevamo
```

Lo stesso succede con l'assegnazione `=`: **non copia l'array**, crea solo un secondo nome per lo stesso array.

```
let a = [1, 2, 3];
let b = a;                                    // b e a sono lo STESSO array
b[0] = 100;
console.log(a);                               // [ 100, 2, 3 ]
```

Per avere una **copia vera** si può usare `let b = [...a];` oppure `let b = a.slice();`.

La nostra regola: una funzione **non modifica mai** l'array che riceve. Se deve produrre un array, ne **costruisce uno nuovo** (partendo da `[]`) e lo **restituisce**. Così l'array di partenza resta com'era e non ci sono sorprese.

```
function raddoppia(v) {
  let risultato = [];                         // array nuovo
  for (let i = 0; i < v.length; i++) {
    risultato[i] = v[i] * 2;
  }
  return risultato;                           // restituito: v non è toccato
}

let a = [1, 2, 3];
console.log(raddoppia(a));                    // [ 2, 4, 6 ]
console.log(a);                               // [ 1, 2, 3 ]
```

Esercizio 5

Ciascun programma ha un **problema**. Dite cosa succede quando lo si esegue (errore? risultato strano?), e correggetelo.

A. Vogliamo stampare tutti gli elementi.

```
let v = [4, 5, 6];
for (let i = 0; i <= v.length; i++) {
  console.log(v[i]);
}
```

B. Vogliamo la somma degli elementi.

```
let v = [4, 5, 6];
let s = 0;
for (let i = 0; i < v.length(); i++) {
  s = s + v[i];
}
console.log(s);
```

C. Vogliamo l'ultimo elemento.

```
let v = [4, 5, 6];
console.log(v[v.length]);
```

D. Vogliamo sommare 1 a ogni elemento.

```
let v = [4, 5, 6];
for (let i = 0; i < v.length; i++) {
  v + 1;
}
console.log(v);
```

E. Vogliamo stampare la somma di tutti gli elementi di `v`.

```
function somma(v) {
  let s = 0;
  for (let i = 0; i < v.length; i++) {
    let s = s + v[i];
  }
  return s;
}
console.log(somma([4, 5, 6]));
```

Esercizio 6

A. Scrivere la funzione `media(v)` che, dato un array di numeri, **restituisce** la media dei suoi elementi. Provarla con `[10, 20, 30]`, `[7]`, `[-4, 4]`.

B. Scrivere la funzione `quantiMaggiori(v, x)` che restituisce **quanti** elementi di `v` sono maggiori di `x`. Ad esempio `quantiMaggiori([3, 9, 5, 12], 4)` restituisce `3`.

Esercizio 7

Scrivere la funzione `posizioneMassimo(v)` che restituisce **l'indice** (non il valore) del massimo. Con `[3, 9, 5, 12, 7]` deve restituire `3`.

Esercizio 8

Scrivere la funzione `posizioni(v, x)` che restituisce un **nuovo array** con **tutti gli indici** in cui compare `x`. Ad esempio `posizioni([0, 5, 0, 0, 7], 0)` restituisce `[0, 2, 3]`; se `x` non compare restituisce `[]`.

Esercizio 9

Come sempre, la funzione **non modifica** l'array che riceve: costruisce e **restituisce un nuovo array**.

Scrivere la funzione `quadrati(v)` che restituisce un nuovo array in cui ogni elemento è il **quadrato** del corrispondente elemento di `v`.

Esercizio 10

Anche qui le funzioni **restituiscono un nuovo array** e non toccano quello ricevuto.

Scrivere la funzione `rimuoviPiccoli(v, n)` che restituisce un nuovo array con gli elementi di `v` **maggiori di** `n`, cioè senza quelli minori o uguali a `n`. Ad esempio `rimuoviPiccoli([5, 1, 8, 3, 9, 2], 3)` restituisce `[5, 8, 9]`.

Esercizio 11

Scrivere la funzione `senzaDuplicati(v)` che restituisce un nuovo array in cui ogni valore compare **una volta sola** (si tiene la prima occorrenza, nello stesso ordine). Ad esempio `senzaDuplicati([4, 7, 4, 9, 7, 4])` restituisce `[4, 7, 9]`. Suggerimento: prima di aggiungere un elemento al risultato, controllate se c'è già.

Esercizio 12

Funzioni che lavorano su **due array**. Come sempre, non modificano gli array ricevuti.

Scrivere la funzione `sommaElementi(a, b)` che, dati due array **della stessa lunghezza**, restituisce un nuovo array in cui l'elemento in posizione `i` è `a[i] + b[i]`. Ad esempio `sommaElementi([1, 2, 3], [10, 20, 30])` restituisce `[11, 22, 33]`.

Esercizio 13

Scrivere la funzione `stessaSomma(a, b)` che restituisce `true` se la somma degli elementi di `a` è uguale alla somma degli elementi di `b` (anche se gli array hanno lunghezze diverse), `false` altrimenti. Suggerimento: riusate la funzione `somma` del Recap 2.

Esercizio 14

Scrivere la funzione `unisci(a, b)` che, dati due array `a` e `b` che rappresentano due insiemi, restituisce l'array che rappresenta l'insieme `a U b`. Ad esempio `unisci([1, 2], [1, 4, 5])` restituisce `[1, 2, 4, 5]`.

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