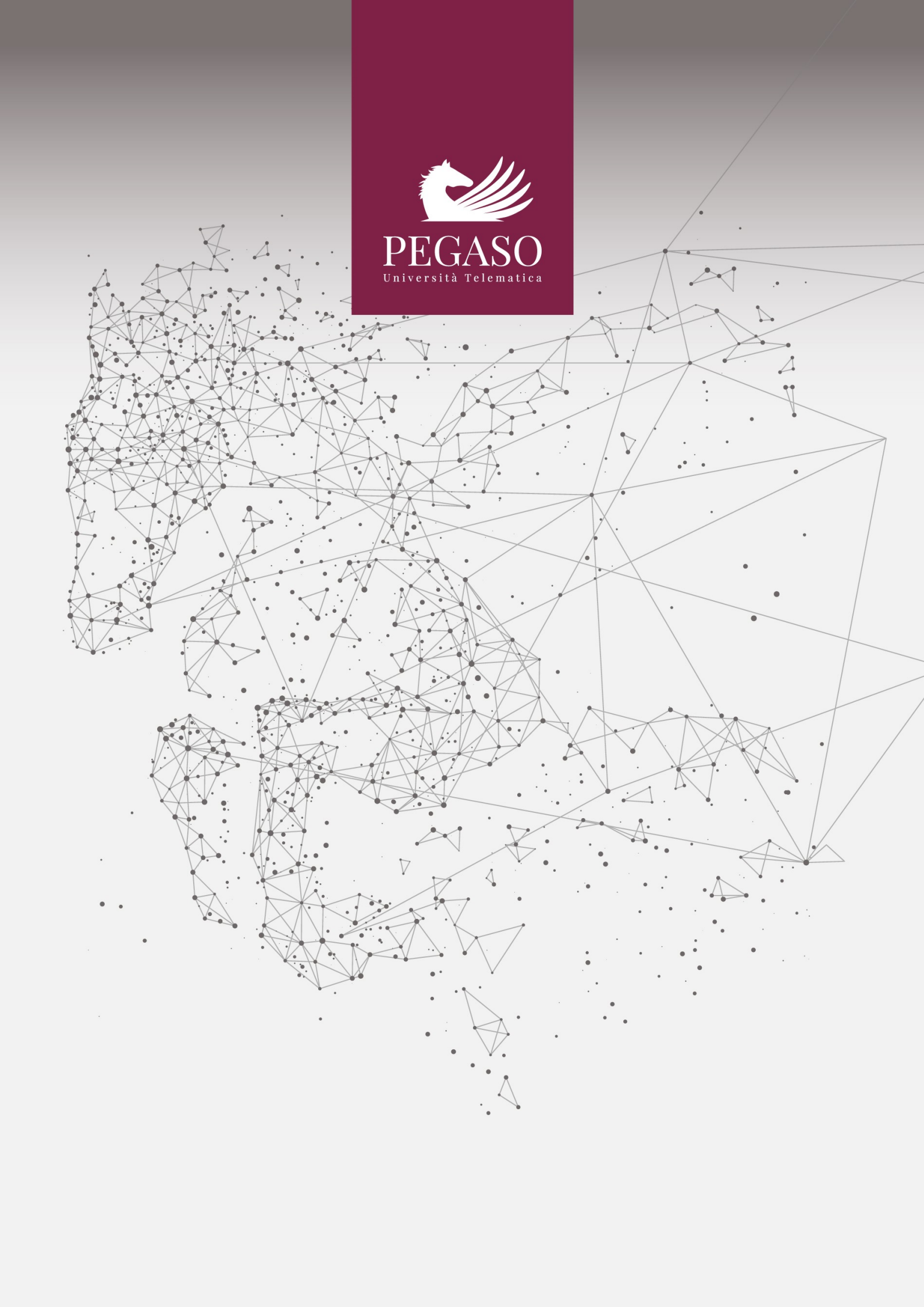
PEGASO
Università Telematica

Indice

1. INTRODUZIONE ALLA STRUTTURA DATI 3

2. IMPLEMENTAZIONE 5

3. C++ ARRAY VS PYTHON LIST 9

BIBLIOGRAFIA E SITOGRAFIA 10

1. Introduzione alla struttura dati

Informalmente, una struttura dati è costituita da uno o più insiemi e da operazioni definite sui loro elementi: questa è una nozione astratta, svincolata dalla concreta rappresentazione della struttura sulla macchina.

Le strutture dati note nell'ambito della programmazione (vettori, record, liste ecc.) possono essere definite ad un livello astratto come insiemi di elementi dotati di opportune operazioni; l'implementazione di una data struttura descrive invece il criterio con il quale i vari elementi sono memorizzati nei registri della macchina e definisce i programmi che eseguono le operazioni.

Ogni struttura dati ammette in generale più implementazioni a ciascuna delle quali corrisponde un costo in termini di spazio, per il mantenimento dei dati in memoria, e uno in termini di tempo, per l'esecuzione dei programmi associati alle operazioni.

Quando parliamo di un array, possiamo immaginare una sorta di contenitore, le cui caselle sono dette celle (o elementi) dell'array stesso. Ciascuna delle celle si comporta come una variabile tradizionale: tutte le celle sono variabili di uno stesso tipo preesistente, detto tipo base dell'array (si parlerà perciò di tipi come "array di interi", "array di stringhe", "array di caratteri" ecc.).

Quello che si ottiene dichiarandolo è dunque un contenitore statico ed omogeneo di valori, variabili o oggetti:

- statico: i suoi elementi non variano di numero a tempo d'esecuzione;
- omogeneo: i suoi elementi sono tutti dello stesso tipo.

Qual è dunque la differenza tra array e variabili? A una variabile può essere assegnato un solo valore mentre ad un array possono essere assegnati più valori.

Figura 1 *- Array*

Ciascuna delle celle dell'array è identificata da un valore di indice. L'indice è generalmente numerico è parte dallo 0: si potrà quindi parlare della cella di indice 0, di indice 1, e, in generale, di indice N, dove N è un intero compreso fra 0 e il valore massimo per gli indici dell'array.

La possibilità di accedere agli elementi attraverso un indice è la principale caratteristica di un array.

È possibile accedere singolarmente ad una sua generica posizione ("accesso casuale", come per la memoria), oltre a scorrerlo sequenzialmente in entrambe le direzioni tramite un ciclo iterativo in tutti i suoi elementi o a partire da alcuni di essi.

Gli elementi di un array sono memorizzati consecutivamente e individuati attraverso la loro posizione: l'indirizzo di ciascuno di essi è determinato dalla somma dell'indirizzo del primo elemento dell'array e dell'indice che individua l'elemento in questione all'interno dell'array.

Poiché l'accesso a ciascun elemento di un array è diretto, l'operazione di lettura o modifica del valore di un elemento di un array ha complessità asintotica $O(1)$ rispetto al numero di elementi dell'array.

Riassumendo:

- L'array è una struttura dati statico ed omogeneo di valori, variabili o oggetti.
- La memoria è allocata immediatamente all'atto di creazione dell'array e resta vuota finché non si assegnano dei valori.
- Gli elementi dell'array sono allocati in posizioni contigue di memoria e possono essere acceduti in maniera efficiente $O(1)$ usando gli indici.
- Gli indici partono dallo 0.

problemi che è interessante analizzare quando si lavora sugli array sono:

- **Problema della visita**: dato un array, attraversare tutti i suoi elementi esattamente una volta.
- **Problema della ricerca**: dati un array e un valore, stabilire se il valore è contenuto in un elemento dell'array, riportando in caso affermativo l'indice di tale elemento.
- **Problema dell'ordinamento**: dato un array i cui elementi contengono tutti una chiave d'ordinamento e data una relazione d'ordine totale sul dominio delle chiavi, determinare una permutazione dei valori contenuti negli elementi dell'array tale che la nuova disposizione delle chiavi soddisfa la relazione.

2. Implementazione

Array in C++

```cpp
#include <iostream>

using namespace std;

int main() {
  char myCharArray[11];
  myCharArray[0]='h';
  myCharArray[1]='e';
  myCharArray[2]='l';
  myCharArray[3]='l';
  myCharArray[4]='o';
  myCharArray[5]=' ';
  myCharArray[6]='w';
  myCharArray[7]='o';
  myCharArray[8]='r';
  myCharArray[9]='l';
  myCharArray[10]='d';

  for (int i=0;i<11;i++)
    cout<<myCharArray[i];
}
```

Line 21 : Col 2

Console × Shell × +

```
sh -c make -s
./main
hello world
```

Figura 2 *- array in c++*

```
char helloWorld[]={'h','e','l','l','o',' ','w','o','r','l','d'};
for (int i=0;i<11;i++)
  cout<<helloWorld[i];
```

Figura 3 *- array in c++*

Array in Python?

Attenzione: Python does not have built-in support for Arrays

Il Python non supporta nativamente gli array, tuttavia è possibile utilizzare le Python Lists che però hanno differenze rispetto al concetto standard di Array: le Python Lists, ad esempio, ammettono dati non omogenei.

Qualora si volessero usare "array" in Python l'ambiente mette a disposizione l'array module oppure la libreria numPy.

Di seguito mostriamo un esempio di creazione di una Python List:

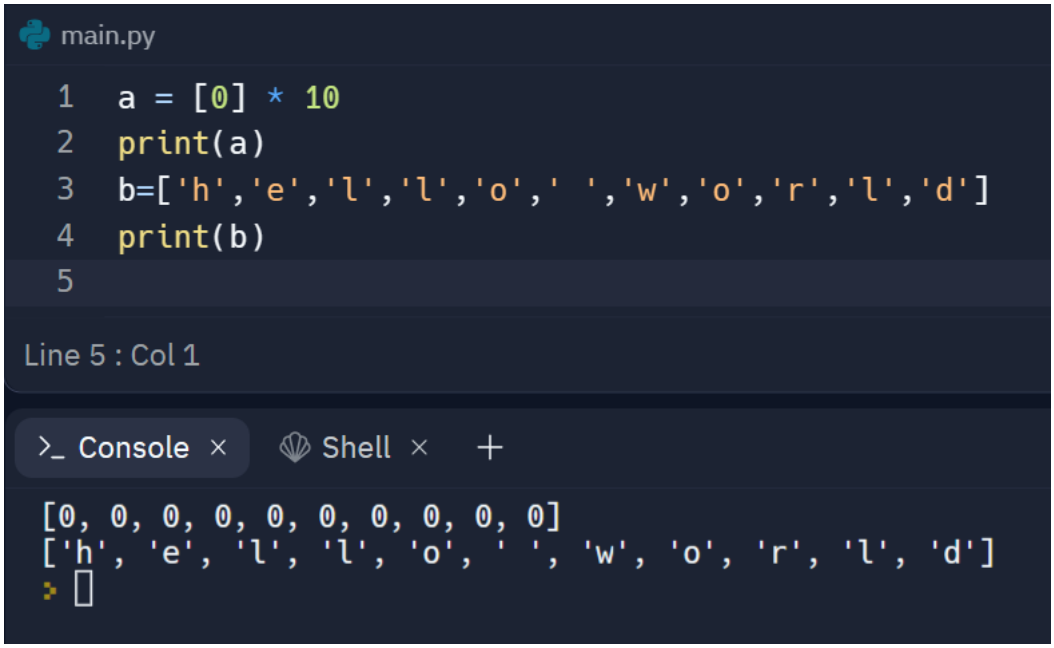

Figura 4 *- Python List*

Di seguito mostriamo invece l'implementazione di un array usando l'array module:

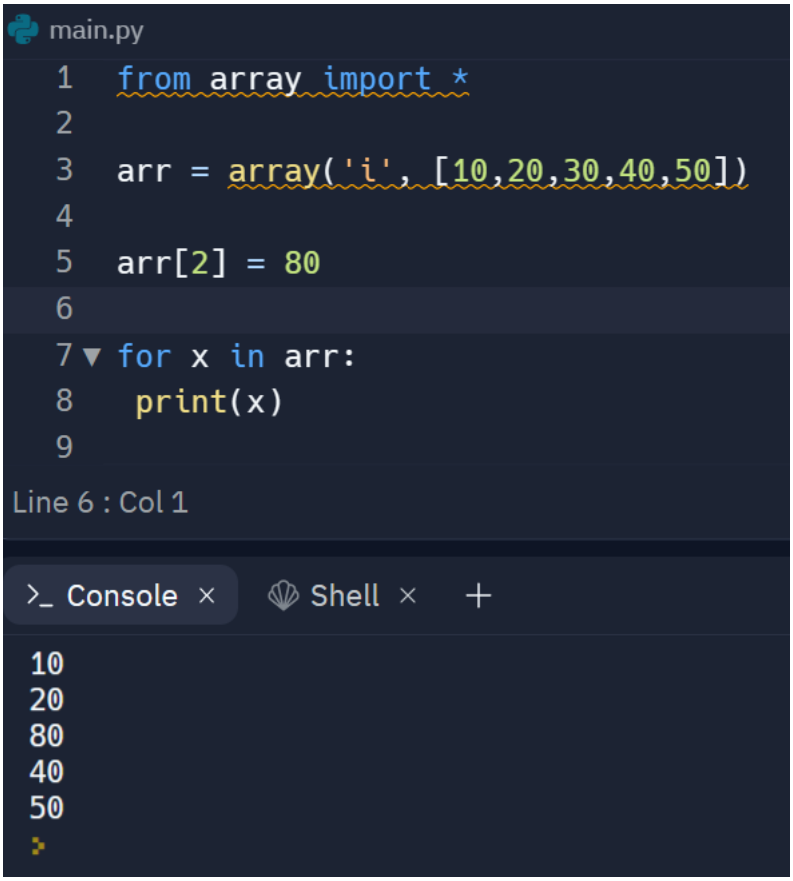

Figura 5 *- array in Python con array module*

Da precisare che a livello "prestazionale", l'array module risulta più "veloce" rispetto alla PythonList.

Array in CIE Pseudocode

È interessante mostrare lo pseudocodice per la creazione di un array secondo lo standard CIE Pseudocode:

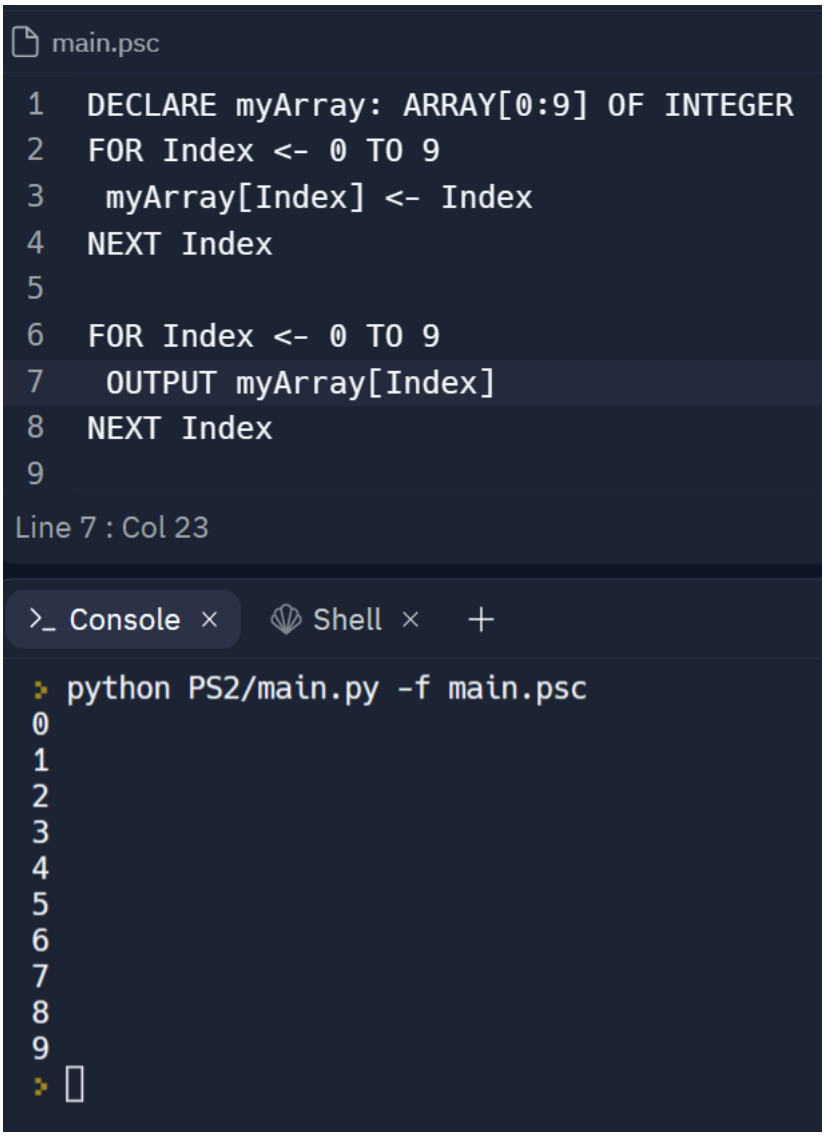

Figura 6 *- array in Pseudocode*

3. C++ array vs Python list

Come già accennato in precedenza, la differenza sostanziale tra array in C++ e Python list è nel fatto che i dati in C++ devono essere omogenei, mentre in una Python list non necessariamente:

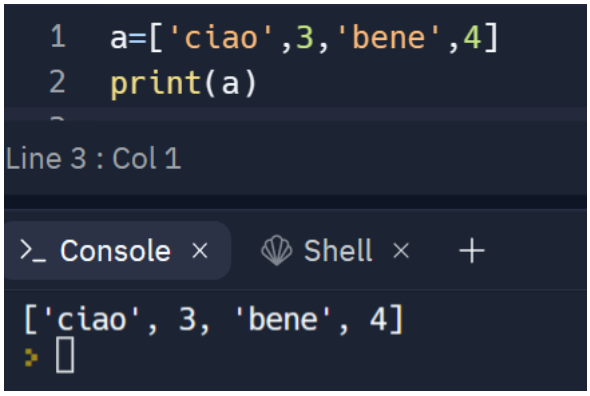

Figura 7 *- Python list non omogenea*

Una discriminante importante sulla scelta tra un array in C++ ed una Python list è la velocità e dunque le prestazioni: l'implementazione dell'array rende il suo utilizzo nettamente più veloce (ricordiamo che le Python list sono implementate utilizzando un array sottostante, fatto di references).

Il fatto che gli elementi di un array sono memorizzati in celle contigue è il motivo per cui le performance di accesso per indice sono molto elevate.

Il Python, tuttavia, offre una protezione che richiede "tempo": in Python non è possibile iterare oltre la fine della lista; in C++ invece ciò è possibile ed è anche consentito cambiare i valori "oltre" la fine dell'array, mettendoci nella condizione di poter generare problemi inattesi.

Bibliografia e sitografia

- Alan Bertossi, Alberto Motresor: Algoritmi e strutture di dati, Città Studi Edizioni, terza edizione
- C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione
- Crescenzi, Gambosi, Grossi: Strutture di Dati e Algoritmi, Pearson/Addison-Wesley
- Sedgewick: Algoritmi in C, Pearson, 2015
- Cormen Leiserson Rivest Stein-Introduzione Agli Algoritmi E Strutture Dati-Prima Edizione
- https://www.andreaminini.com/