

Laboratorio 5

Elementi di Informatica e Programmazione

Esercizio 1: lettura di un file CSV

I file `csv` (Comma Separated Values) sono un comunissimo formato di scambio dati in formato tabella. La loro diffusione è dovuta principalmente alla loro semplicità. Una tabella viene rappresentata in un file CSV con una riga di testo per ogni riga di tabella: le colonne all'interno di una riga sono separate da una virgola. Spesso, la prima riga di un file `csv` è l'intestazione, ovvero contiene i nomi delle colonne.

Ad esempio un file `csv` con questo contenuto

```
Cognome,Nome,Codice Fiscale,Giorni ricovero,Priorità
Burattini,Francesca,BRTFNC75E54L781J,2,A
Beruto,Piergiorgio,BRTPGR46T10A059G,3,B
Carlassare,Noemi,CRLNMO61M45L736M,1,P
```

corrisponde a questa tabella

| Cognome | Nome | Codice Fiscale | Giorni Ricovero | Priorità |
|------------|-------------|------------------|-----------------|----------|
| Burattini | Francesca | BRTFNC75E54L781J | 2 | A |
| Beruto | Piergiorgio | BRTPGR46T10A059G | 3 | B |
| Carlassare | Noemi | CRLNMO61M45L736M | 1 | P |

In questo esercizio dobbiamo implementare una funzione `read_csv` che riceve un solo parametro e deve ritornare *una lista di dizionari* con un dizionario per ogni riga della tabella. Le chiavi del dizionario sono le intestazioni di colonna che compaiono nella prima riga.

Il file dell'esempio è quindi rappresentato dalla seguente lista di dizionari:

[

```
[
    {'Cognome': 'Burattini', 'Nome': 'Francesca', 'Codice Fiscale': 'BRTFNC75E54L781J',
     'Giorni ricovero': '2', 'Priorità': 'A'},
    {'Cognome': 'Beruto', 'Nome': 'Piergiorgio', 'Codice Fiscale': 'BRTPGR46T10A059G',
     'Giorni ricovero': '3', 'Priorità': 'B'},
    {'Cognome': 'Carlassare', 'Nome': 'Noemi', 'Codice Fiscale': 'CRLNMO61M45L736M',
     'Giorni ricovero': '1', 'Priorità': 'P'},
]
```

Per svolgere l'esercizio seguite i seguenti passi:

- Aprite il file usando le procedure standard utilizzate anche in altri esercizi
- Leggete *solo la prima riga* e la trasformate in una lista di nomi di colonna dividendola sul carattere `,`. A questo scopo potete usare il metodo `s.split(",")` delle stringhe.
- Leggete le restanti righe, tagliando le stringhe sulle virgole e creando i dizionari corrispondenti prendendo le chiavi dalle intestazioni lette nella prima riga e i valori dalle posizioni appropriate nella riga corrente

La firma della funzione è la seguente:

```
def read_csv(path):
    # scrivi qui
```

Esercizio 2 - Massimo Comun Divisore

Progettare una funzione che

- riceve due numeri interi (positivi) come parametri
- calcola e restituisce il massimo comune divisore (M.C.D.) dei due numeri ricevuti

Si ricorda che il M.C.D. di due numeri interi positivi è il più grande numero intero che è divisore di entrambi. Per calcolare il M.C.D. di due numeri, `m` e `n`, si implementi la seguente procedura:

- se `n` è un divisore di `m`, allora `n` è il loro M.C.D.
- altrimenti, il M.C.D. di `m` e `n` è uguale al M.C.D. di `n` e del resto della

divisione intera di `m` per `n`. Inserire la funzione nel programma `recursiveMCD.py` che contenga anche la funzione `main`, che chiede all'utente di fornire due numeri interi positivi e ne visualizza il M.C.D.

Esercizio 3 - sequenza di Fibonacci

Scrivere il programma `recursiveAndIterativeFibonacci.py` che contenga due funzioni

- `recursiveFib`
- `iterativeFib`

Entrambe ricevono come parametro un numero intero non negativo, n , e restituiscono l' n -esimo numero `Fib(n)` nella sequenza di Fibonacci, così definita:

- $\text{Fib}(0) = 0$
- $\text{Fib}(1) = 1$
- $\text{Fib}(n) = \text{Fib}(n-2) + \text{Fib}(n-1)$ per ogni $n > 1$

La funzione `recursiveFib` calcola il valore da restituire usando la ricorsione doppia, implementando direttamente la formula, mentre la funzione `iterativeFib` deve calcolare il valore da restituire senza usare la ricorsione e senza usare strutture dati di memorizzazione (cioè senza liste/tuple, ma usando soltanto variabili “semplici”).

Inserire nel programma la funzione `main` che:

- chieda all'utente il numero n desiderato
- calcoli `Fib(n)` invocando `iterativeFib`, misurando il impiegato per il calcolo usando la funzione `perf_counter` del modulo `time`, illustrata nel seguito
- calcoli `Fib(n)` invocando `recursiveFib`, misurando il impiegato per il calcolo usando la funzione `perf_counter`
- visualizzi `Fib(n)` (oppure un messaggio d'errore se i due valori calcolati sono diversi...)
- visualizzi il tempo impiegato dalla funzione `iterativeFib`
- visualizzi il tempo impiegato dalla funzione `recursiveFib`

Osservare come, all'aumentare di n , il tempo impiegato dalla funzione iterativa rimanga pressoché trascurabile, mentre il tempo impiegato dalla funzione ricorsiva aumenti in modo esponenziale, seguendo approssimativamente la funzione 2^n , cioè il tempo richiesto per calcolare `Fib(n)` è circa il doppio di quello richiesto per calcolare `Fib(n-1)`.

Ricordate che per misurare il tempo di esecuzione di una frammento di codice potete usare il codice seguente

```
# All'inizio del file
import time

start = time.time()
# codice cronometrato
# ...
end = time.time()
```

```
elapsed = end - start # tempo trascorso, in secondi
```