Il file `prenotazioni.csv` contiene una riga di intestazione e le prenotazioni delle stanze di un hotel per il mese di agosto 2026. L'hotel è composto da 10 piani, ciascuno dotato di 5 stanze (numerate da 1 a 5 all'interno del piano), per un totale di 50 stanze. Ciascuna riga del file contiene quattro valori separati da virgola: il numero di piano, il numero di stanza all'interno del piano, il giorno di inizio (check-in) e il giorno di fine (check-out) della prenotazione (giorni di agosto numerati da 1 a 31, estremi inclusi). Di seguito si riporta un estratto del file.

1: File prenotazioni.csv

```
piano,stanza,checkin,checkout
1,1,3,5
2,5,1,4
...
10,5,2,8
```

Le prenotazioni vengono gestite tramite una matrice booleana di 50 righe e 31 colonne: l'elemento di indice [i][j] vale True se la stanza di indice i è occupata nel giorno di indice j di agosto (j=0 corrisponde al 1° agosto), False altrimenti. L'indice della stanza è calcolato a partire dal piano e dal numero di stanza all'interno del piano (i=0 corrisponde alla stanza numero 1 del piano 1, i=5 corrisponde alla stanza numero 1 del piano 2, i=49 corrisponde alla stanza numero 5 del piano 10). La seguente tabella riporta, a titolo d'esempio, una rappresentazione leggibile di un estratto della matrice di occupazione dell'hotel (T=True, F=False). Per brevità sono mostrati solo alcuni giorni e alcuni piani. Nella tabella sono cerchiate le celle corrispondenti alle prenotazioni riportate nell'estratto del file precedente.

Piano	Stanza	Giorni di agosto														
		1	2	3	4	5	…	21	22	23	…	27	28	29	30	31
	1	F	F	(T)	(T)	(T)	…	F	F	T	…	F	F	T	F	F
	2	T	T	F	F	F	…	F	T	T	…	F	F	F	T	T
1	3	F	T	T	F	F	…	T	T	F	…	F	T	T	F	F
	4	F	F	F	T	T	…	F	F	F	…	T	T	F	F	T
	5	T	F	F	F	T	…	T	F	F	…	F	F	T	T	F
	1	F	T	T	F	F	…	F	F	T	…	T	F	F	F	T
	2	T	T	F	F	T	…	F	T	F	…	F	F	T	T	F
2	3	F	F	T	T	F	…	T	T	T	…	F	F	F	T	T
	4	T	F	F	T	T	…	F	F	F	…	T	T	F	F	F
	5	(T)	(T)	(T)	(T)	F	…	F	T	F	…	F	F	T	F	F
							…									
	1	F	T	T	F	F	…	F	F	T	…	T	T	F	F	F
	2	T	F	F	T	T	…	T	T	F	…	F	F	T	F	F
10	3	F	F	T	T	F	…	F	T	T	…	T	F	F	F	T
	4	T	T	F	F	F	…	F	F	F	…	F	T	T	T	F
	5	F	(T)	(T)	(T)	(T)	…	F	F	T	…	T	T	F	F	F

Table 1: Rappresentazione della matrice di occupazione di un hotel di 50 stanze per 31 giorni. Si noti che le colonne `Piano` e `Stanza`, così come le intestazioni dei giorni, non fanno parte della matrice: servono solamente a rendere leggibile l'estratto riportato.

Si assuma che nel programma siano definite le seguenti costanti: `PIANO_MIN = 1`, `PIANO_MAX = 10`, `GIORNO_MIN = 1`, `GIORNO_MAX = 31`, `STANZE_PER_PIANO = 5`.

1. Definire la funzione `leggi_prenotazioni`

   - **Parametri di ingresso**:
     - `percorso_file`: stringa
   - **Restituisce**: una lista di tuple, ciascuna costituita da quattro interi.
   - **Descrizione**: la funzione apre il file il cui percorso è fornito nel parametro di ingresso, salta la riga di intestazione, e costruisce per ciascuna delle righe rimanenti una tupla contenente i quattro valori interi riportati nella riga. Restituisce la lista di tali tuple.
   - **Esempio**: se la funzione viene chiamata con il percorso del file `prenotazioni.csv` come argomento, essa restituisce la seguente lista (troncata per ragioni di spazio):
     `[(1,1,3,5), (2,5,1,4), ..., (10,5,2,8)]`

2. Definire la funzione `inserisci_prenotazione`

   - **Parametri di ingresso**:
     - `matrice`: lista di liste di valori booleani
     - `piano`: intero
     - `stanza`: intero
     - `checkin`: intero
     - `checkout`: intero
   - **Restituisce**: un valore booleano.
   - **Descrizione**: la funzione calcola l'indice di riga della matrice in base ai valori di `piano` e `stanza`, e verifica se esiste almeno un valore True tra gli elementi i cui indici corrispondono ai giorni da `checkin` a `checkout`, estremi inclusi. In tal caso la funzione non modifica la `matrice` e restituisce False. Se invece tutti gli elementi valgono False, modifica la `matrice` impostandoli a True, e restituisce True.
   - **Esempio**: con riferimento alla matrice rappresentata in Tabella 1, si riportano due esempi di valori degli altri parametri di ingresso e il valore restituito dalla funzione:
     - `piano=1, stanza=2, checkin=2, checkout=5` → `False` (il giorno 2 è già prenotato)
     - `piano=10, stanza=3, checkin=28, checkout=30` → `True` (i tre elementi corrispondenti sono False; la matrice viene modificata inserendo la prenotazione)

3. Definire la funzione `riempi_matrice`

   - **Parametri di ingresso**:
     - `lista_prenotazioni`: lista di tuple di quattro interi
   - **Restituisce**: lista di liste di booleani
   - **Descrizione**: la funzione crea una matrice 50×31 inizializzata a False. Successivamente considera, una alla volta, le tuple presenti nella `lista_prenotazioni` ricevuta in ingresso e invoca la funzione `inserisci_prenotazione` per aggiornare la matrice in base agli intervalli indicati. La funzione restituisce infine la matrice ottenuta. Si può assumere che la lista in ingresso sia consistente: i valori presenti nelle tuple sono validi e non ci sono sovrapposizioni.
   - **Esempio**: con riferimento all'estratto del file riportato nella pagina iniziale, la lista delle prenotazioni contiene, tra le altre, le tuple `(1,1,3,5)`, `(2,5,1,4)` e `(10,5,2,8)`. Tali tuple sono riportate soltanto come esempio: la funzione deve elaborare tutte le prenotazioni contenute nella lista ricevuta in ingresso, non solo quelle mostrate nell'estratto. La matrice restituita deve quindi contenere il valore `True` in tutte le celle corrispondenti alle prenotazioni presenti nella lista.

4. Infine, definire la funzione `main` in modo che svolga i seguenti compiti:
   - ottenga la lista delle prenotazioni, invocando la funzione `leggi_prenotazioni` con argomento il percorso `'prenotazioni.csv'`;
   - crei e riempia la matrice booleana delle prenotazioni, invocando la funzione `riempi_matrice`, specificando come argomento la lista di prenotazioni acquisita nel punto precedente;
   - permetta all'utente di effettuare una nuova richiesta di prenotazione:
     - acquisisca da tastiera due interi che rappresentano il giorno di check-in e il giorno di check-out;
     - se i valori non sono validi, stampi un opportuno messaggio di errore e permetta di reinserire i valori fino all'inserimento di valori validi;
     - verifichi se esiste almeno una stanza disponibile per tutto l'intervallo richiesto;
       * se esiste, stampi piano e numero di stanza della prima stanza disponibile ed inserisca la relativa prenotazione;
       * se non esiste alcuna stanza disponibile, stampi un opportuno messaggio.

Legenda

`s`, `s1`: stringa `a`, `b`: numero `i`, `j`, `k`, `n`: intero `x`: elemento generico
`l`, `l1`: lista `d`, `d1`: dizionario `r`, `r1`: set `t`, `t1`: tupla

Principali funzioni built-in (ordine alfabetico)

`abs(a)`: restituisce il valore assoluto di un numero

`enumerate(iterable)`: restituisce un iteratore i cui elementi sono tuple contenenti il conteggio e il valore ottenuto dall'iterazione su iterable

`input(s)`: scrive il prompt `s` sullo standard output e restituisce stringa acquisita

`isinstance(object, classinfo)`: restituisce True se `object` è del tipo `classinfo` o di una sottoclasse di `classinfo`. Altrimenti, restituisce False

`len(x)`: restituisce la lunghezza (numero di elementi) di `x`. `x` può essere una sequenza (ad es.: string, list, tuple, range) o un contenitore (ad es.: dict, set)

`max(iterable, *, key=None)`, `max(arg1, arg2, *args, key=None)`: restituisce il massimo nell'iterabile, o il massimo tra due o più argomenti. La funzione `key` serve a calcolare il valore da usare per confrontare gli elementi

`min`: analogo a `max`

`print(*x, sep=' ', end='\n')`: stampa oggetti, separati da `sep` e seguiti infine da `end`

`range(j)`, `range(i, j, k)`: restituisce una sequenza immutable per progressione aritmetica

`reversed(seq)`: restituisce un iteratore che percorre `seq` in ordine inverso

`round(a, n)`: arrotonda il valore di `a` all'intero più vicino o ad `n` cifre decimali

`sorted(iterable, key=None, reverse=False)`: restituisce una nuova lista ordinata degli elementi di `iterable`. La funzione `key` serve a calcolare il valore da usare per confrontare gli elementi. `reverse=True` inverte l'ordine

`sum(iterable)`: restituisce la somma dei valori dell'iterabile

`zip(*iterables)`: itera in parallelo su diversi iterabili e restituisce un iteratore costituito da tuple. La i-esima tupla è costituita dall'i-esimo elemento di ciascuno degli iterabili

Indexing e Slicing

`seq[i]`: restituisce l'elemento di indice `i` della sequenza

`seq[i:j]`: restituisce una sottosequenza di `seq` dall'indice `i` all'indice `j` (escluso)

`seq[i:j:k]`: restituisce una sottosequenza di `seq` da indice `i` a indice `j` (escluso) con step `k`

Modulo Random

`choice(seq)`: restituisce un elemento casuale dalla sequenza `seq`

`choices(seq, k=1)`: restituisce una lista di `k` elementi scelti da `seq` con reinserimento

`random()`: restituisce un float casuale in [0,1)

`randint(i, j)`: restituisce un intero casuale tra `i` e `j` (inclusi)

`sample(seq, k)`: restituisce una lista di `k` elementi unici scelti da `seq` senza reinserimento

`shuffle(seq)`: rimescolamento della sequenza `seq` in-place

`uniform(a, b)`: restituisce un float casuale tra `a` e `b` (inclusi)

Principali funzioni del modulo math

`math.cos(a)`, `math.sin(a)`, `math.exp(a)`, `math.log(a)`, `math.log2(a)`, `math.sqrt(a)`

Principali eccezioni

`FileNotFoundError`, `IndexError`, `KeyError`, `ValueError`, `Exception` (eccezione generica)

Stringhe e principali metodi

`str()`, **Literal:** `"abc"` o `'abc'`

Trasformazioni (non in-place): `s.lower()`, `s.upper()`, `s.replace(s1,s2)`, `s.strip()`
Controlli: `s.startswith(s1)`, `s.endswith(s1)`, `s.islower()`, `s.isupper()`, `s.isdigit()`
Da lista a stringa: `s.join(l)`, Da stringa a lista: `s.split(sep)`
Altro: `s.count(s1)`, `s.index(s1)`; Conversione: a intero `int(s)`, a float `float(s)`
Concatenazione: `s1 + s2` Replicazione: `s1 * n` Membership: `s in s1`

Liste e principali metodi

`list()`, **Literal:** `[x1, x2, ...]`

Modifiche (in-place): `l.append(x)`, `l.extend(l1)`, `l.insert(i, x)`, `l.pop(i)`, `l.remove(x)`, `l.reverse()`, `l.sort(key=None, reverse=False)`
Altro: `l.count(x)`, `l.index(x)`
List comprehension: `l = [`*expression* `for` *item* `in` *iterable*`]`
Concatenazione: `l1 + l2` Replicazione: `l1 * n` Membership: `x in l`

Tuple e principali metodi

`tuple()`, **Literal:** `(x1, x2, ...)`

Altro: `t.count(x)`, `t.index(x)`
Concatenazione: `t1 + t2` Replicazione: `t1 * n` Membership: `x in t`

Insiemi e principali metodi

`set()`, **Literal:** `{x1, x2, ...}`

Modifiche (in-place): `r.add(x)`, `r.remove(x)`, `r.discard(x)`, `r.clear()`
Operazioni: `r.difference(r1)` (oppure `r - r1`), `r.symmetric_difference(r1)` (oppure `r^r1`), `r.union(r1)` (oppure `r|r1`), `r.intersection(r1)` (oppure `r&r1`)
Controlli: `r.issubset(r1)`, `r.issuperset(r1)`, `r.isdisjoint(r1)`
Membership: `x in r1`

Dizionari e principali metodi

`dict()`, **Literal:** `{k1:v1, ...}`

Modifiche (in-place): `d1.update(d2)`, `d.pop(key)`
Viste: `d.keys()`, `d.values()`, `d.items()`
Accesso: `d[key]`, `d.get(key, x=None)`
Aggiunta o modifica: `d[key] = val`
Membership: `key in d1`
Dict comprehension: `d = {`*key:val* `for` *item* `in` *iterable*`}`

f-string

`f"abc {`*expression*`:`*format*`}"`

Esempio numeri (cifre decimali): `f"{3.14:.1f}"` → `'3.1'`
Esempio stringa con allineamento (`<`, `>`, `^`): `f"{'ciao':<10}"` → `'ciao      '`

```
f"{'Trieste':<10} {10:>3}"   → 'Trieste    10'
f"{'Udine':<10} {7:>3}"      → 'Udine      7'
```

File

Apertura: `open(file, mode, encoding)` apre un file e restituisce un file object `f`
Chiusura: `f.close()`; automatica se il file è aperto con il `with` statement
Lettura: `f.read()` (restituisce una stringa con l'intero contenuto del file), `f.readline()` (restituisce una stringa, fino al prossimo \n), `f.readlines()` (restituisce una lista di stringhe)
Scrittura: `f.write(s)`, `f.writelines(l)`