

14 – Queues

Una **queue** (coda) è una struttura dati di tipo **FIFO** (*First In, First Out*), cioè il primo elemento inserito è il primo ad essere rimosso. In Python, si può implementare una queue con **liste** o, più efficientemente, con `collections.deque`.

Operazioni principali

Le operazioni fondamentali di una queue sono:

1. **Enqueue** – aggiungere un elemento in fondo alla coda.
2. **Dequeue** – rimuovere l'elemento in testa.
3. **Peek/Front** – leggere l'elemento in testa senza rimuoverlo.
4. **IsEmpty** – verificare se la coda è vuota.

Esempio con lista (meno efficiente)

```
queue = []

# Enqueue
queue.append(10)
queue.append(20)
queue.append(30)
print(queue) # [10, 20, 30]

# Dequeue
primo = queue.pop(0)
print(primo) # 10
print(queue) # [20, 30]

# Peek
cima = queue[0]
print(cima) # 20
```

Nota: rimuovere il primo elemento con `pop(0)` ha complessità $O(n)$, quindi non è ottimale per code grandi.

Esempio con `collections.deque` (più efficiente)

```
from collections import deque

queue = deque()

# Enqueue
queue.append(10)
queue.append(20)
queue.append(30)

# Dequeue
primo = queue.popleft()
print(primo) # 10
print(queue) # deque([20, 30])

# Peek
cima = queue[0]
print(cima) # 20
```

`deque` permette operazioni di aggiunta e rimozione da entrambi i lati in **O(1)**.

Usi comuni delle queue

- Gestione di **code di processi** o task.
- Algoritmi **BFS** (Breadth-First Search) su grafi e alberi.
- Streaming di dati o buffer temporanei.
- Implementazione di **scheduler** e code di eventi.

Riassunto

- Queue = FIFO, stack = LIFO.
- Liste possono funzionare come queue, ma `deque` è più efficiente.
- Operazioni principali: `enqueue` (append), `dequeue` (pop o `popleft`) e `peek`.

Vuoi procedere con il **paragrafo 15 – Tuples?**