Intelligenza Artificiale

Anno Accademico 2022 - 2023

Strutture Dati in Python:

Liste Ordinate, Pile, Code

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Sommario

- Liste Ordinate (Sorted Lists)
- Pile
- Code

Liste Ordinate
Introduzione

Una *Lista Ordinata* è una lista i cui elementi sono ordinati secondo una certa chiave.

Ogni operazione di inserimento di un nuovo elemento deve dunque rispettare l'ordinamento suddetto.

Vediamo qui di seguito come possiamo realizzare una Lista Ordinata mediante liste concatenate in Python.

Liste ordinate
Definizione della Classe Nodo

```
class Node:
    data = None
    next = None

    def __init__(self, data):
        self.data = data
```

Istanze di questa Classe sono oggetti **`Node`** con attributo **`data`** impostato dal costruttore con il valore del parametro passato.

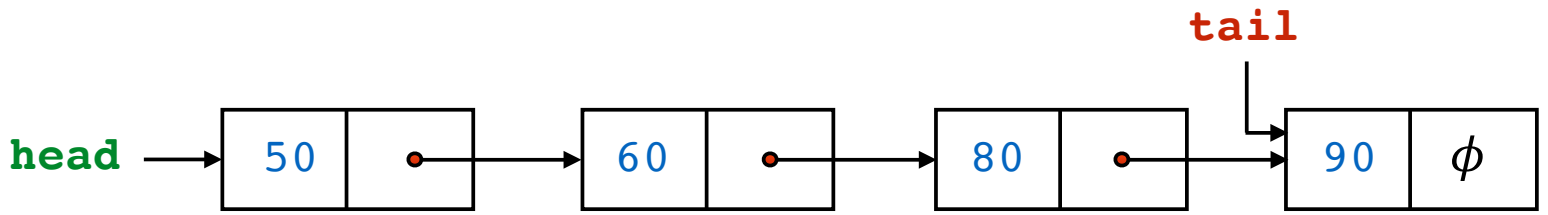
Liste ordinate
Definizione della Classe Lista

```
class SortedLink:
#     __head = None
#     __tail = None

    def __init__(self):
        self.__head = None
        self.__tail = None
```

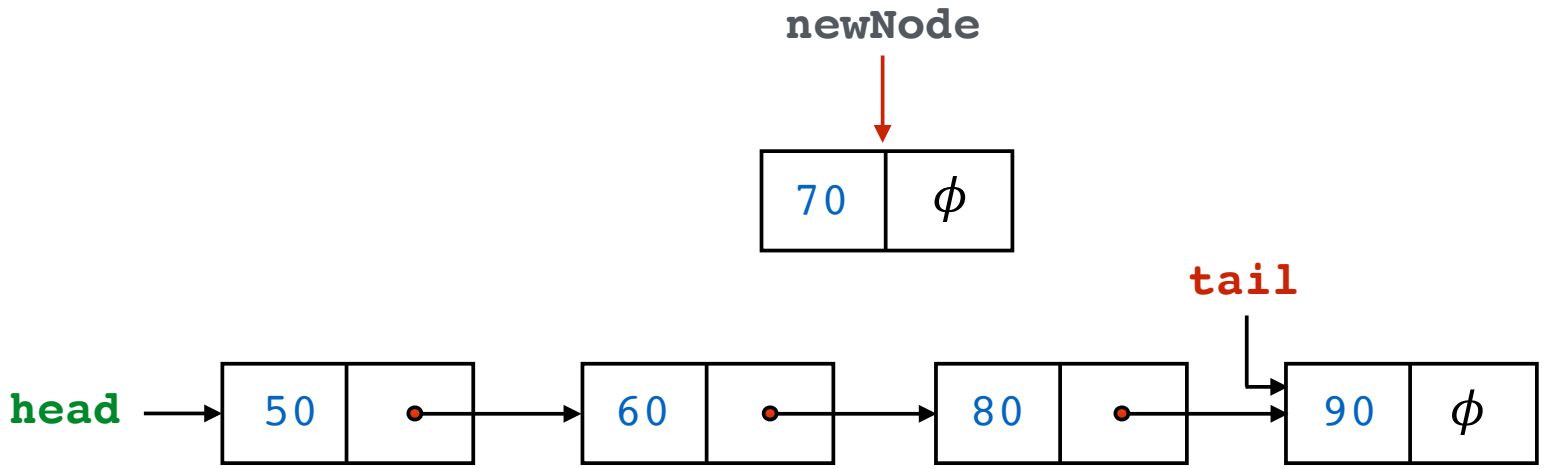
Istanze di questa Classe sono oggetti **SortedList** con i puntatori **head** e **tail** impostati a **None**. Sono quindi inizializzati a liste vuote.

Liste Ordinate
Inserimento in posizione intermedia

Lista di partenza:

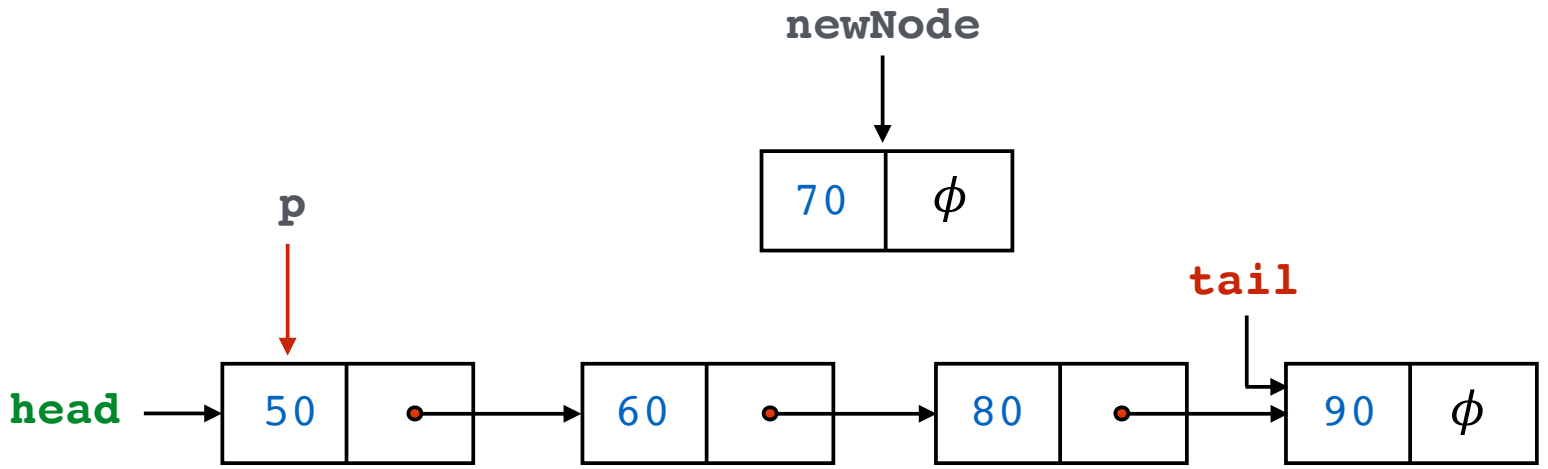


Liste Ordinate
Inserimento in posizione intermedia



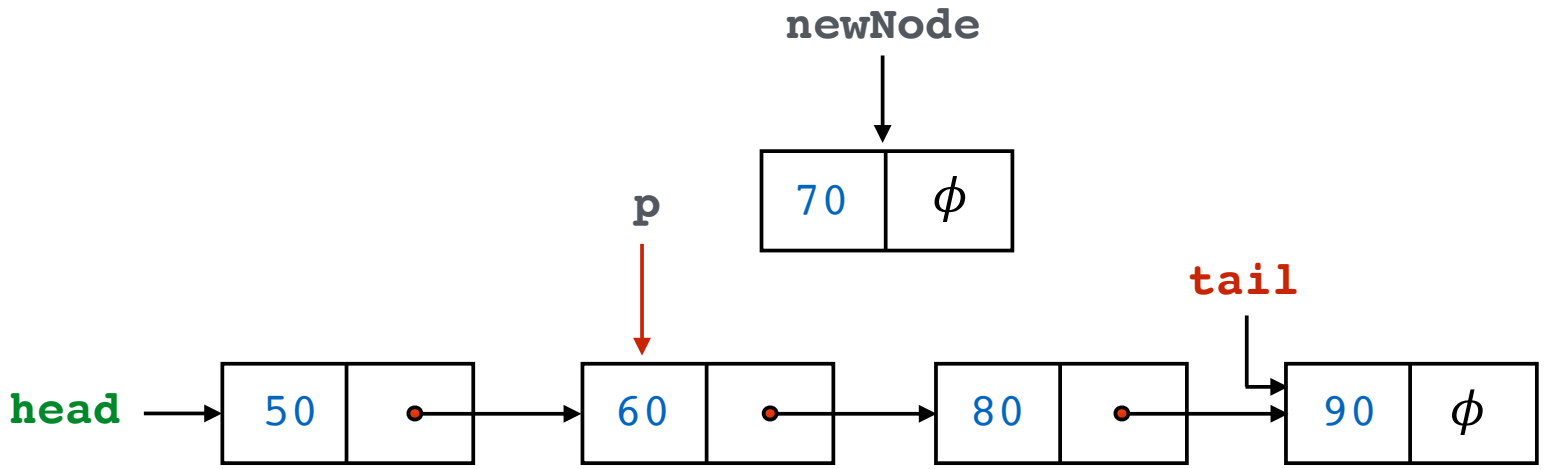
Istruzione: `Creazione del nodo da inserire`

Liste Ordinate
Inserimento in posizione intermedia



Istruzione: `p = self.__head`

Liste Ordinate
Inserimento in posizione intermedia

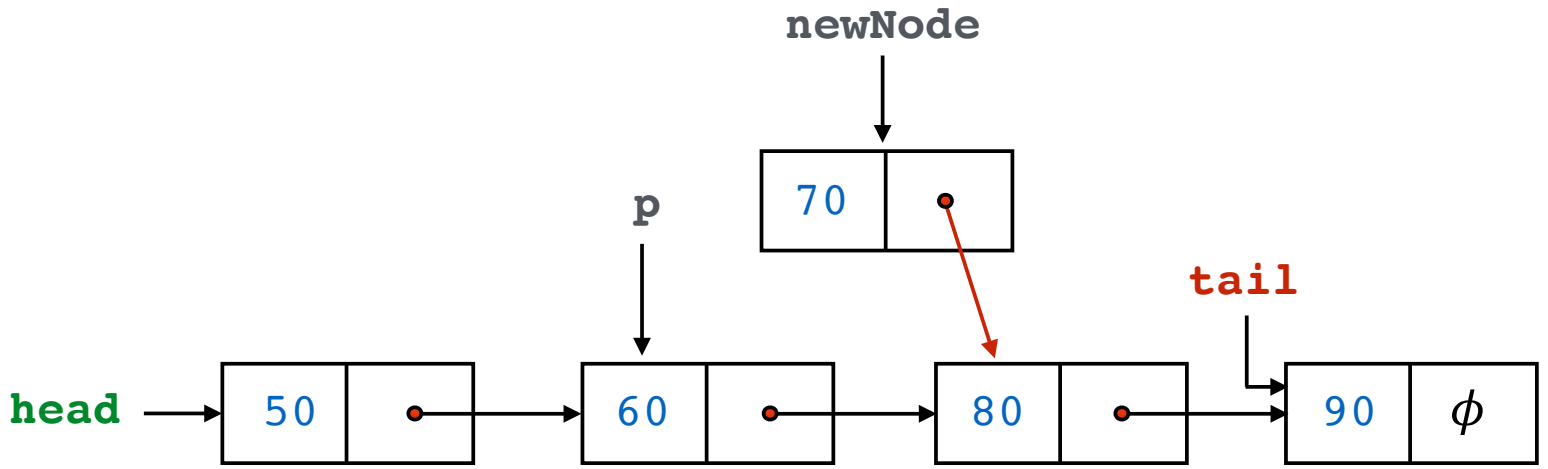


Istruzione: `p = p.next`

Si procede fino a quando vale la condizione:

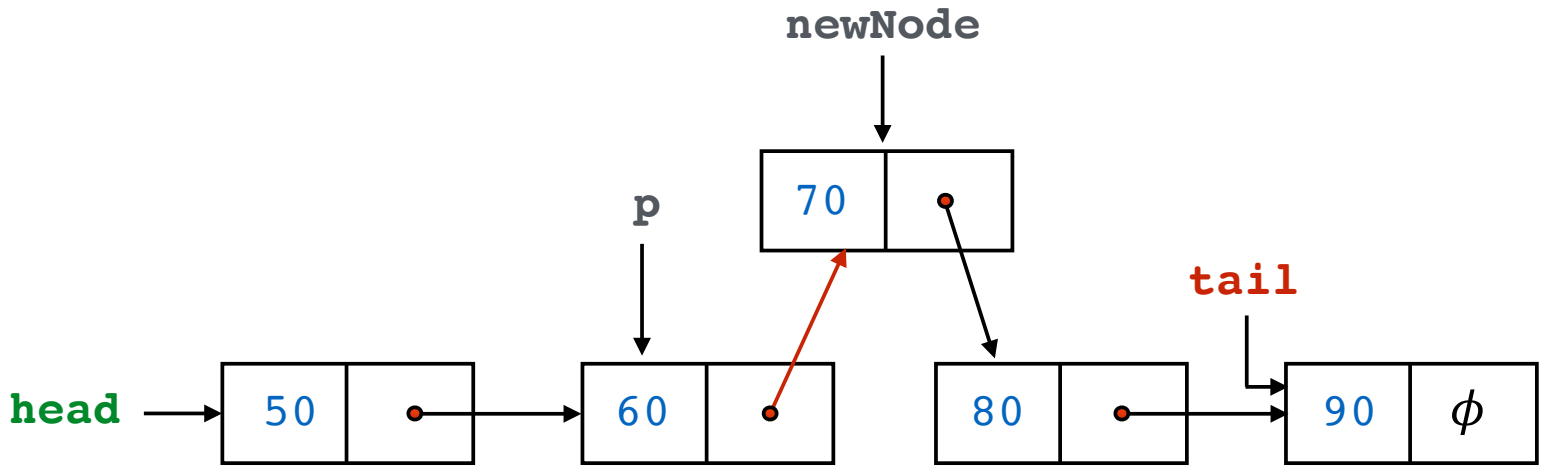
`newNode.data > p.next.data`

Liste Ordinate
Inserimento in posizione intermedia



Istruzione: `newNode.next = p.next`

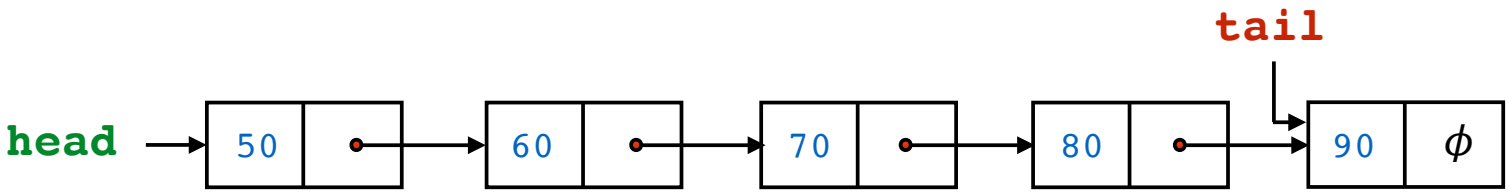
Liste Ordinate
Inserimento in posizione intermedia



Istruzione: `p.next = newNode`

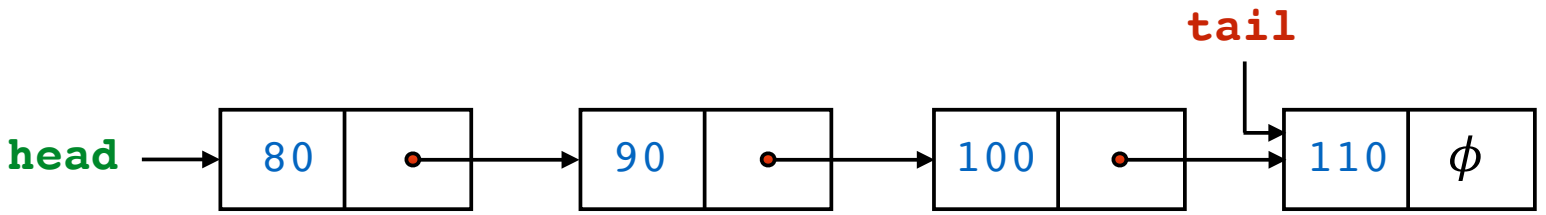
Liste Ordinate
Inserimento in posizione intermedia

Risultato finale:

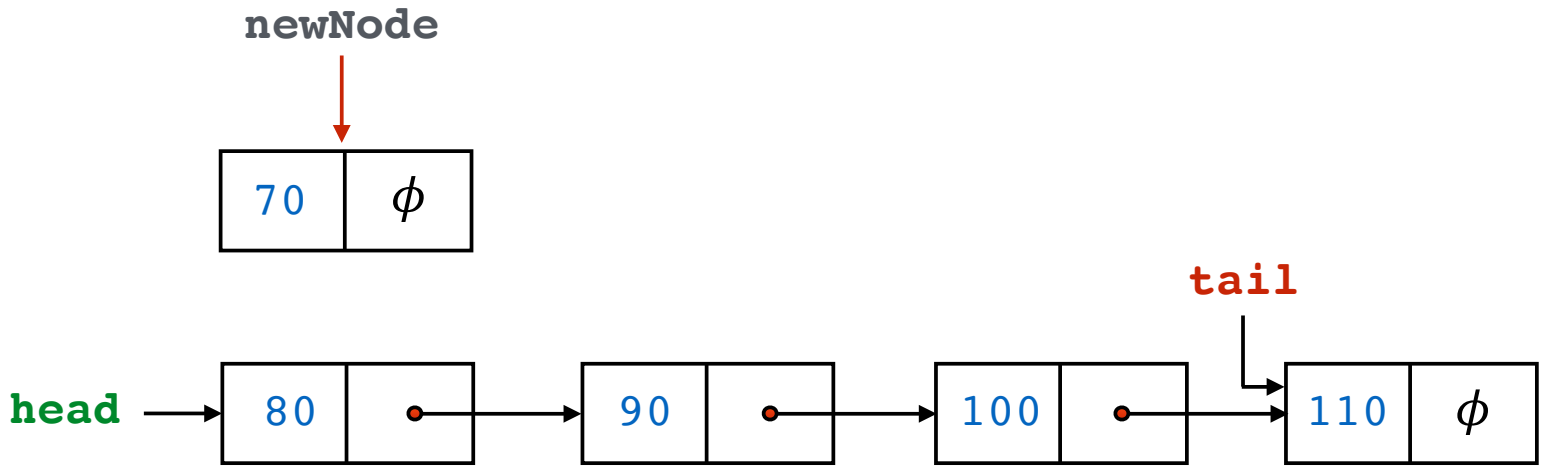


Liste Ordinate
Inserimento in testa

Lista di partenza:

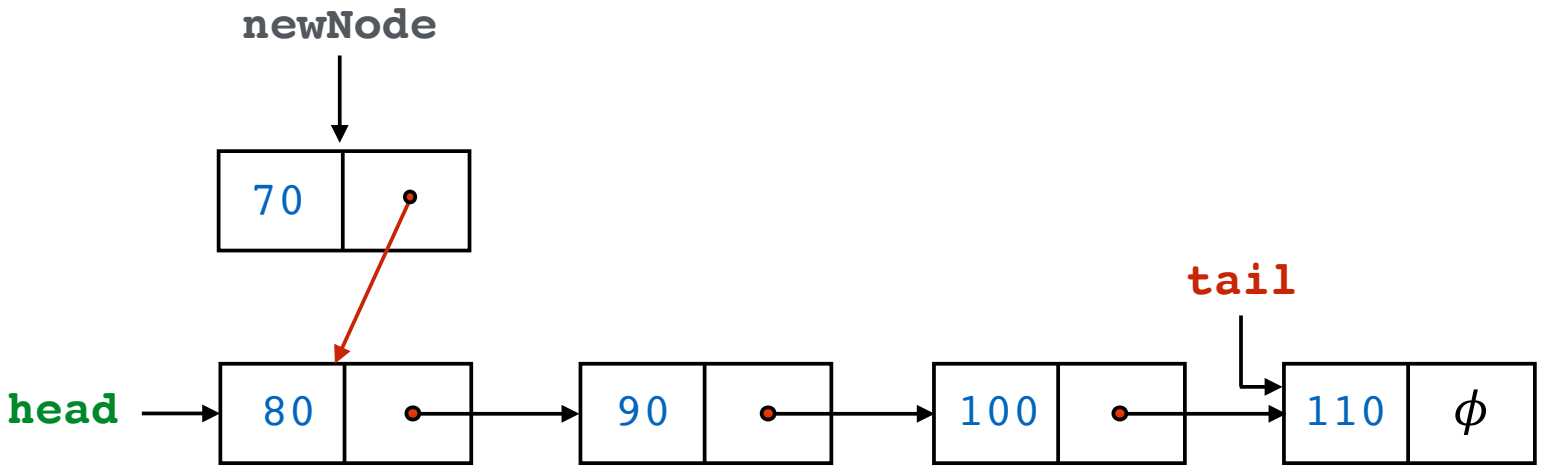


Liste Ordinate
Inserimento in testa



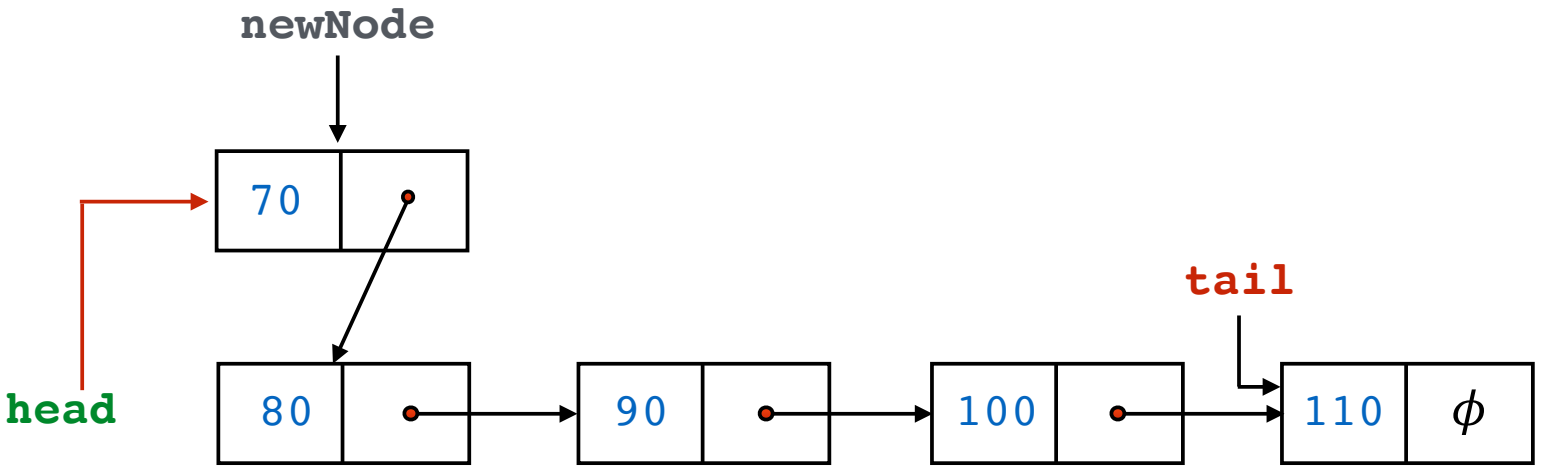
Istruzione: `Creazione del nodo da inserire`

Liste Ordinate
Inserimento in testa



Istruzione: `newNode.next = self.__head`

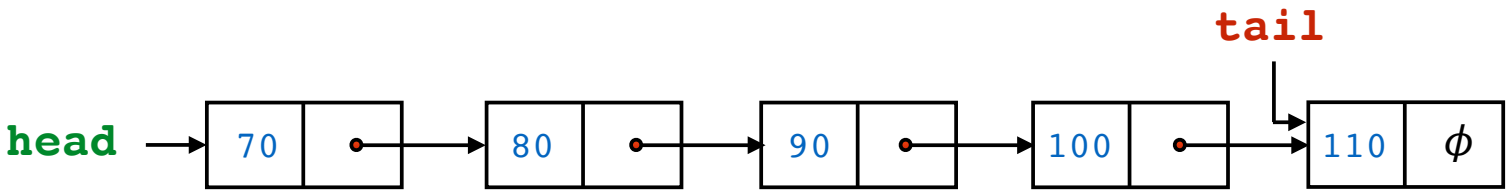
Liste Ordinate
Inserimento in testa



Istruzione: `self.__head = newNode`

Liste Ordinate
Inserimento in testa

Risultato finale:

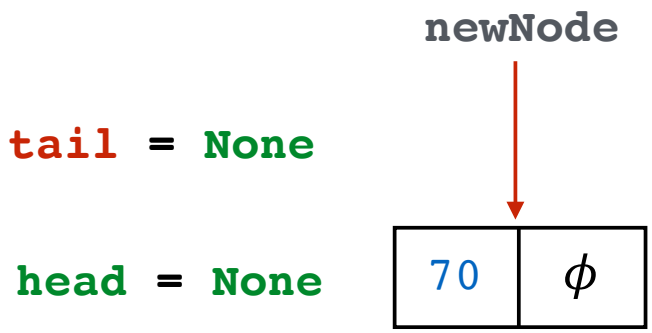


Liste Ordinate
Inserimento in caso di lista vuota

Lista di partenza:

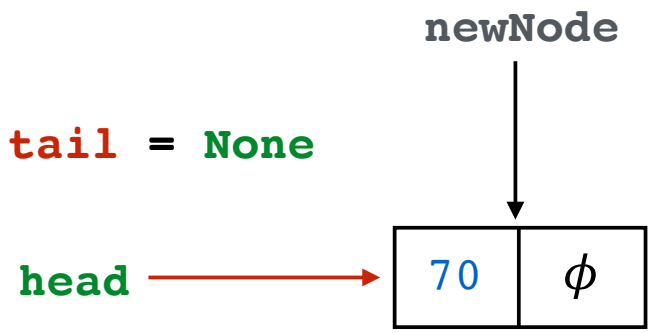
```
tail = None

head = None
```

Liste Ordinate
Inserimento in caso di lista vuota



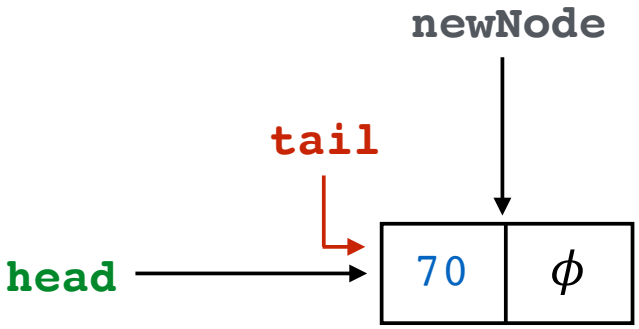
Istruzione: `Creazione del nodo da inserire`

Liste Ordinate
Inserimento in caso di lista vuota



Istruzione: `self.__head = newNode`

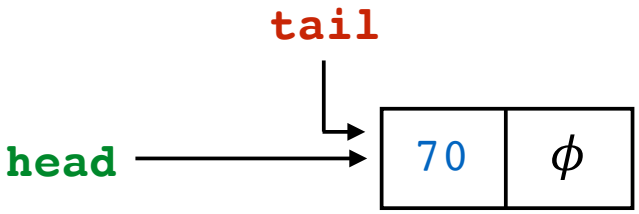
Liste Ordinate
Inserimento in caso di lista vuota



Istruzione: `self.__tail = self.__head`

Liste Ordinate
Inserimento in caso di lista vuota

Risultato finale:



Liste Ordinate
Inserimento di un elemento

Codice del metodo **add**:

```
def add(self, newNode):
    p = self.__head
    if (self.__head == None):                  # se la lista è vuota ...
        self.__head = newNode                  # inserisci l'elemento
        self.__tail = self.__head
        newNode.next = None

    elif (newNode.data > self.__tail.data):    # se il valore è maggiore dell'ultimo ...
        self.__tail.next = newNode             # fai una append
        self.__tail = newNode
        newNode.next = None

    elif (newNode.data < self.__head.data):    # se è minore del primo ...
        newNode.next = self.__head             # inserisci in testa
        self.__head = newNode

    else:
        while(p.next != None and (newNode.data > p.next.data)):  # scandisci la lista
            p = p.next                                           # fino al punto di inserimento
        newNode.next = p.next
        p.next = newNode
```

Pile
Introduzione

Una *pila* è un tipo di dato astratto, basato sul modello dei dati delle liste, in cui tutte le operazioni vengono effettuate a un estremo della lista, chiamato la *testa* (`top`) della pila.

La politica di gestione è la Last-In First-Out (LIFO).

Vediamo qui di seguito come possiamo realizzare una pila mediante liste concatenate in Python.

Pile
Definizione della Classe Nodo

```
class Node:
    data = ''
    next = None

    def __init__(self, data):
        self.data = data
```

Istanze di questa Classe sono oggetti **`Node`** con attributo **`data`** impostato dal costruttore con il valore del parametro passato.

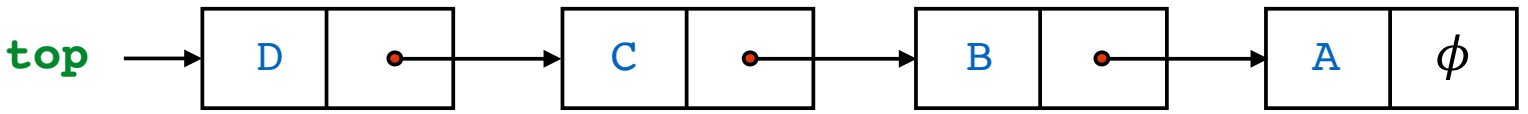
Pile
Definizione della Classe Stack

```
class Stack:
    __top = None
    __size = 0
```

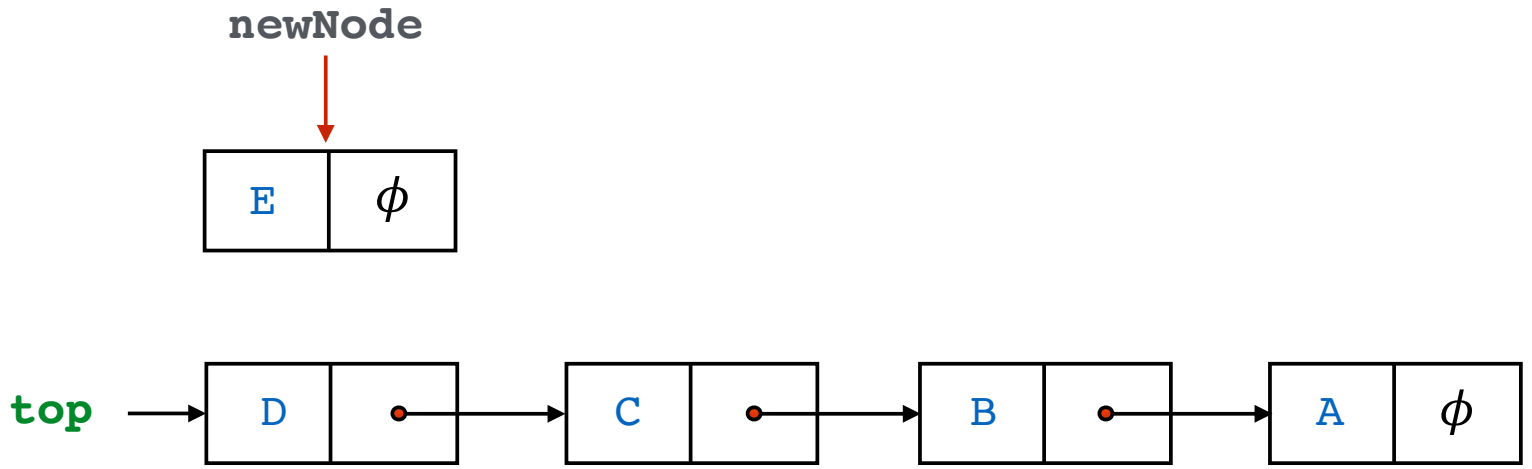
Istanze di questa Classe sono oggetti **`Stack`** con il puntatore **`top`** impostato a **`None`**. Sono quindi inizializzati a liste vuote.

Pile
Inserimento di un elemento (push)

Pila di partenza:

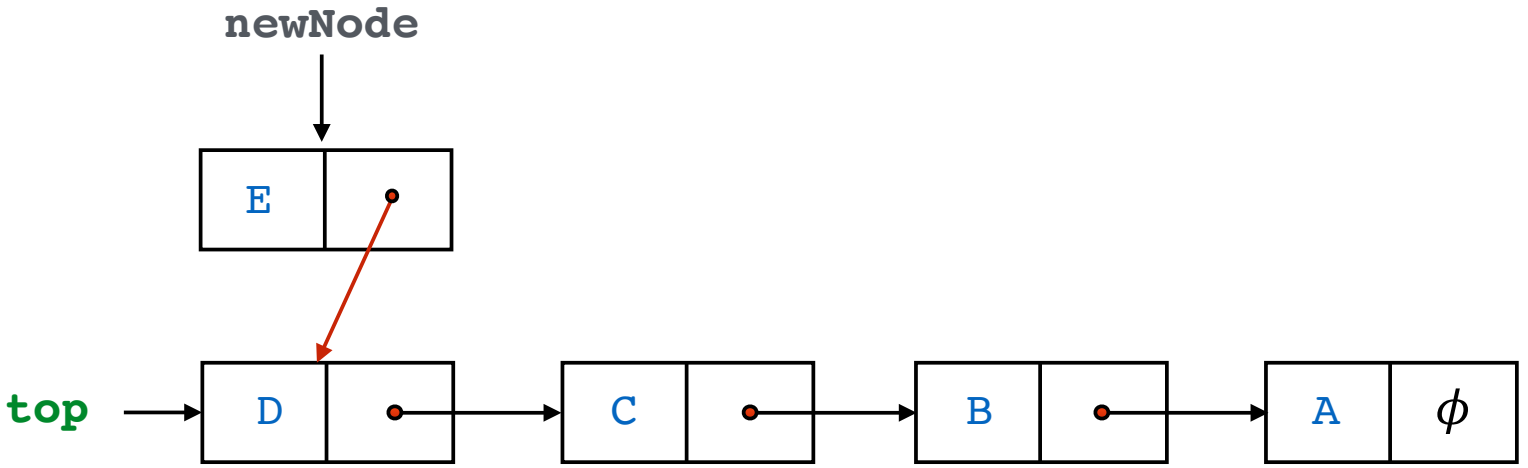


Pile
Inserimento di un elemento (push)



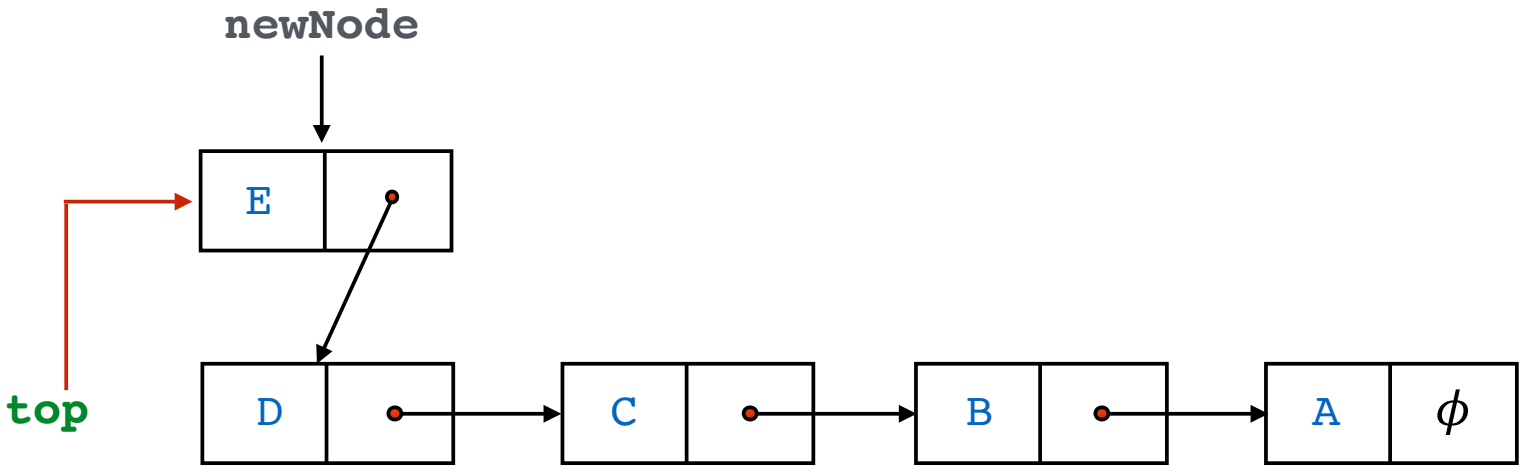
Istruzione: `Creazione del nuovo nodo da inserire`

Pile
Inserimento di un elemento (push)



Istruzione: `newNode.next = self.__top`

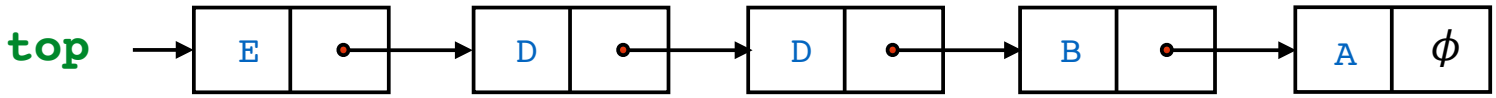
Pile
Inserimento di un elemento (push)



Istruzione: `self.__top = newNode`

Pile
Inserimento di un elemento (push)

Risultato finale:



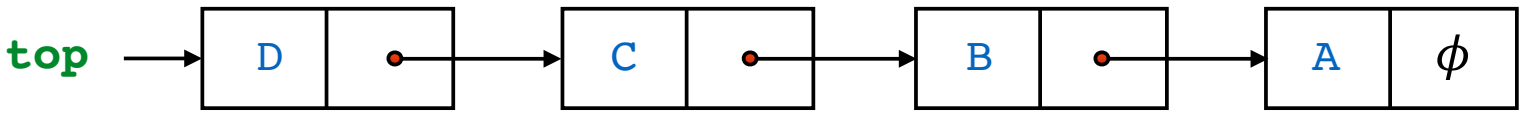
Pile
Inserimento di un elemento (push)

Codice del metodo **push**:

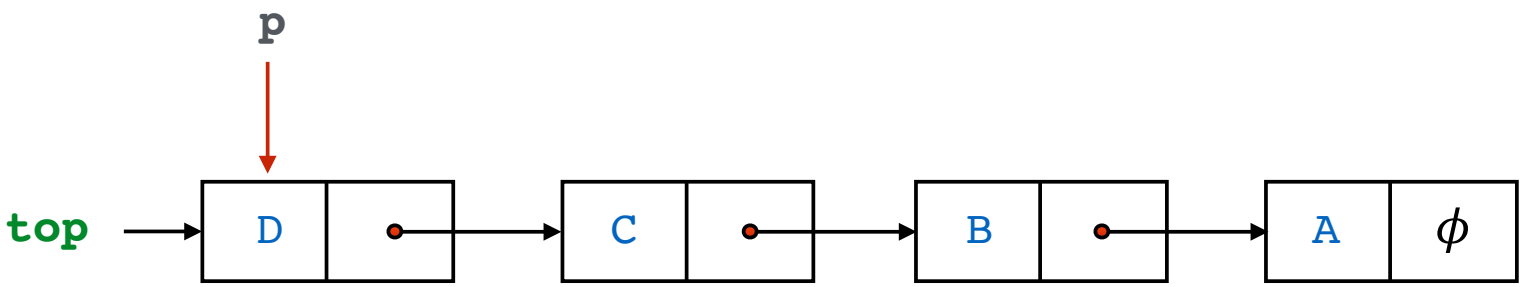
```
def push(self, element):
    if self.__top == None:
        self.__top = Node(element)
    else:
        newNode = Node(element)
        newNode.next = self.__top
        self.__top = newNode
    self.__size += 1
```

Pile
Estrazione di un elemento (pop)

Pila di partenza:

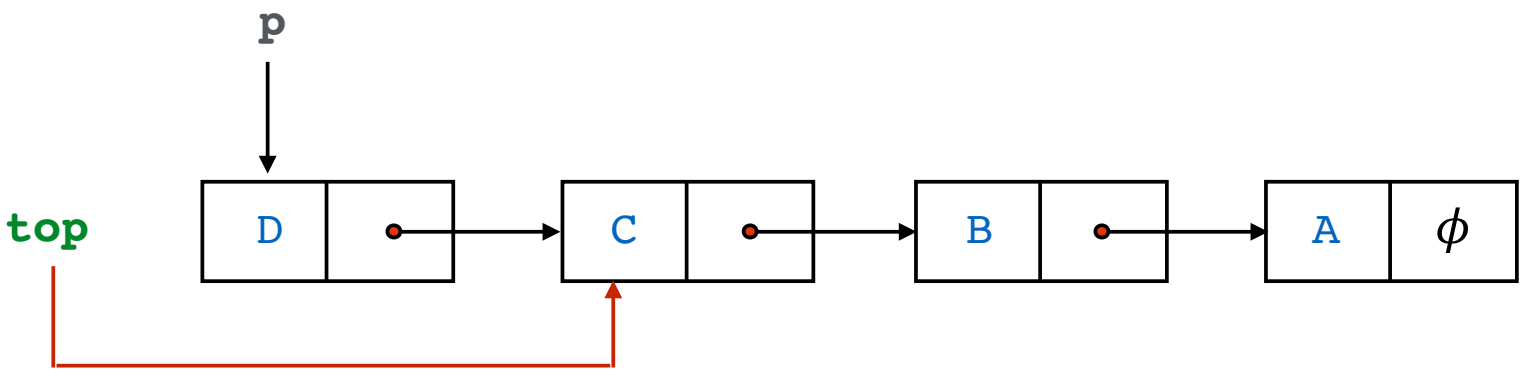


Pile
Estrazione di un elemento (pop)



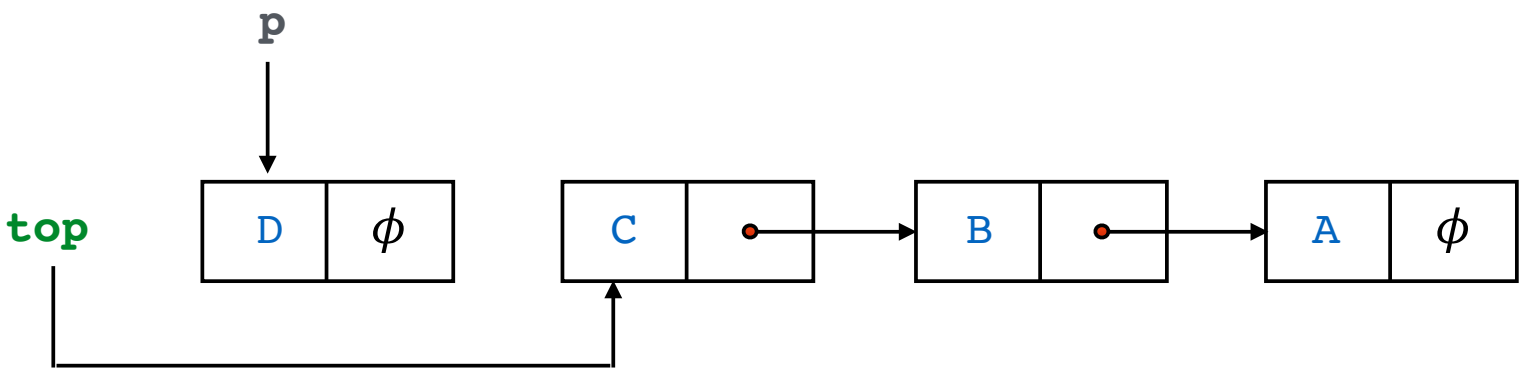
Istruzione: `p = self.__top`

Pile
Estrazione di un elemento (pop)



Istruzione: `self.__top = self.__top.next`

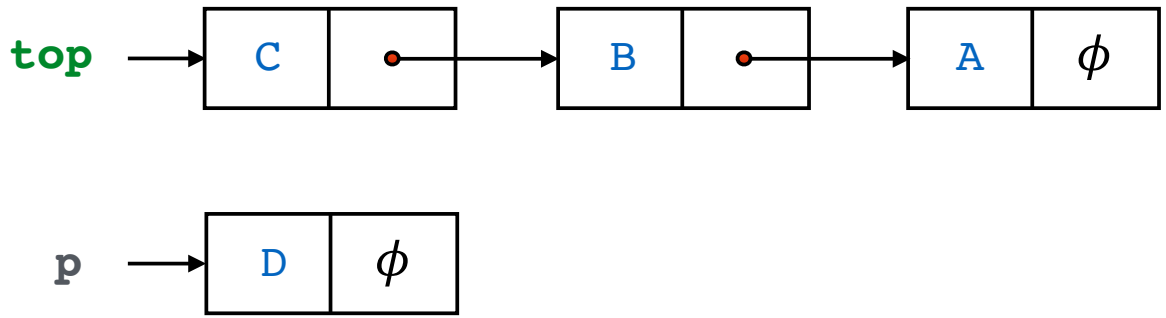
Pile
Estrazione di un elemento (pop)



Istruzione: `p.next = None`

Pile
Estrazione di un elemento (pop)

Risultato finale:



Pile
Estrazione di un elemento (pop)

Codice del metodo **pop**:

```
def pop(self):
    if self.__top == None:
        return None
    p = self.__top
    self.__top = self.__top.next
    p.next = None
    self.__size -= 1
    return p
```

Code
Introduzione

Un altro importante tipo di dato astratto basato sul modello dei dati lista è la *coda*, una forma ristretta di lista in cui gli elementi vengono inseriti a un estremo, il *retro* (`tail`), ed estratti all'altro estremo, la *fronte* o *testa* (`head`).

La politica di gestione è dunque la First-In First-Out (FIFO).

Vediamo qui di seguito come possiamo realizzare una coda mediante liste concatenate in Python.

Code
Definizione della Classe Nodo

```
class Node:
    data = ''
    next = None

    def __init__(self, data):
        self.data = data
```

Istanze di questa Classe sono oggetti **`Node`** con attributo **`data`** impostato dal costruttore con il valore del parametro passato.

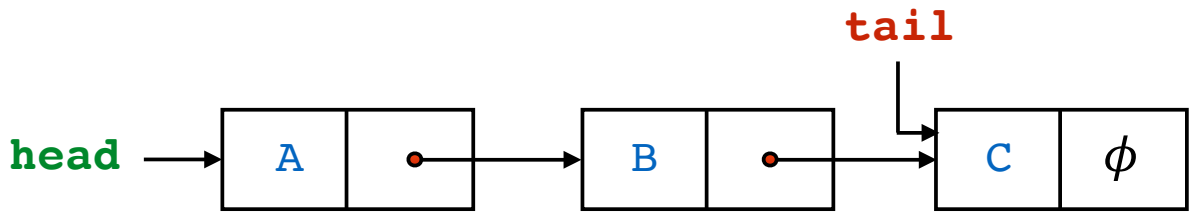
Code
Definizione della Classe Queue

```
class Queue:
    __head = None
    __tail = None
    __size = 0
```

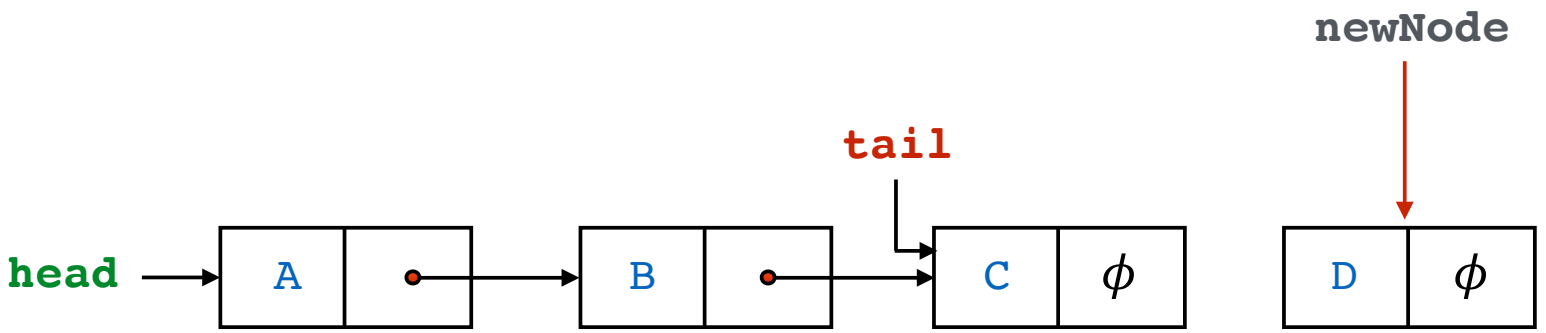
Istanze di questa Classe sono oggetti **`Queue`** con i puntatori **`head`** e **`tail`** impostati a **`None`**. Sono quindi inizializzati a liste vuote. L'attributo **`size`** dovrà contenere il numero degli elementi della coda.

Code
Inserimento di un elemento (enqueue)

Coda di partenza:

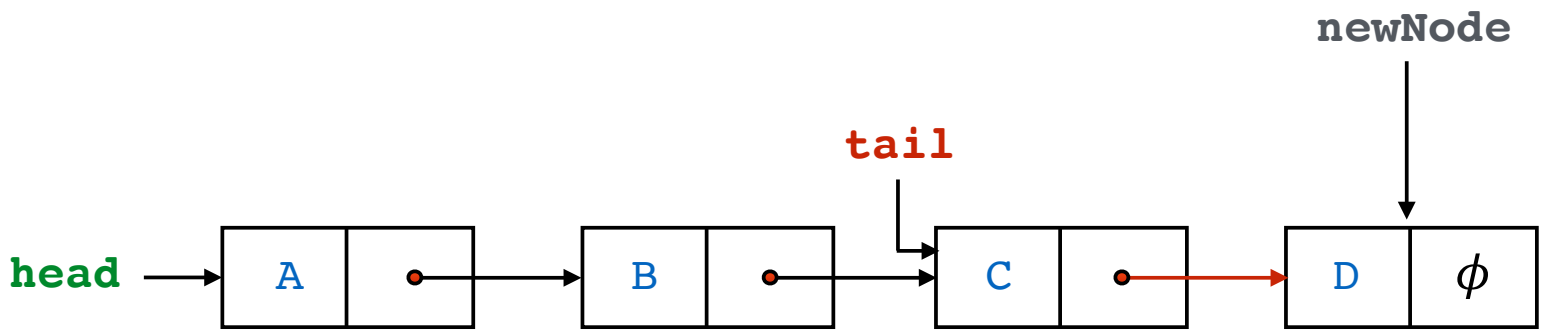


Code
Inserimento di un elemento (enqueue)



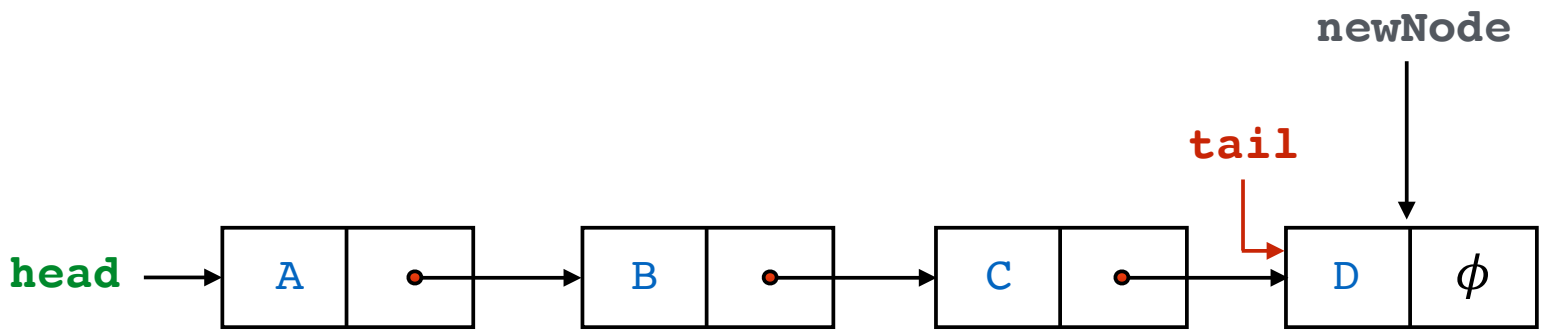
Istruzione: `Creazione dell'oggetto da inserire`

Code
Inserimento di un elemento (enqueue)



Istruzione: `self.__tail.next = newNode`

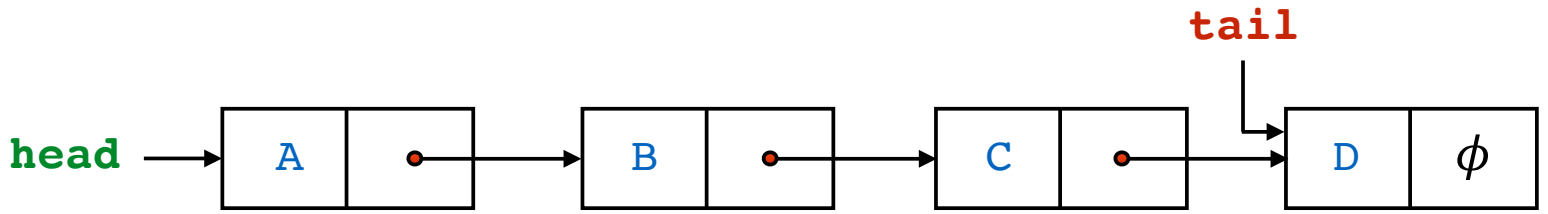
Code
Inserimento di un elemento (enqueue)



Istruzione: `self.__tail = newNode`

Code
Inserimento di un elemento (enqueue)

Risultato finale:



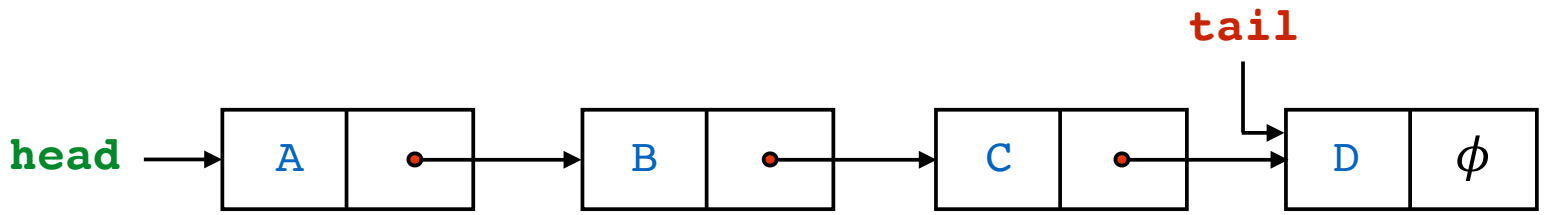
Code
Inserimento di un elemento (enqueue)

Codice del metodo **enqueue**:

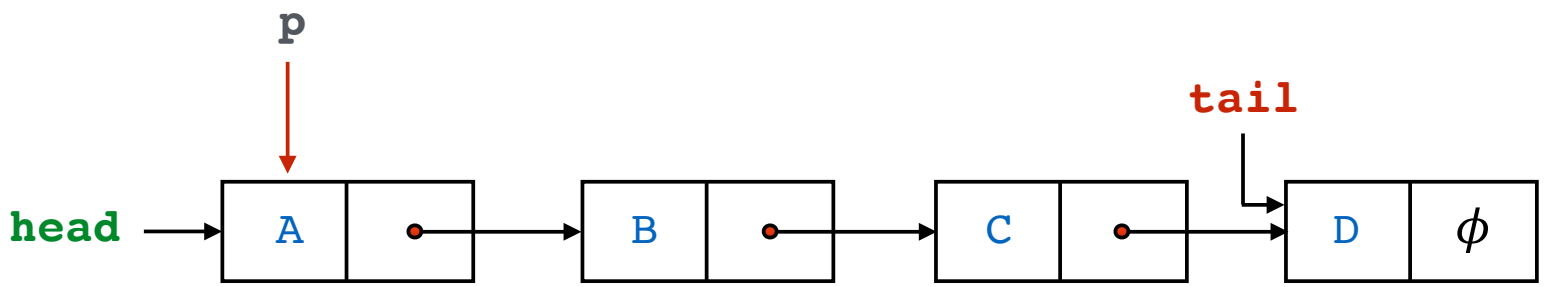
```
def enqueue(self, element):
    newNode = Node(element)
    if self.__head == None:
        self.__head = newNode
        self.__tail = self.__head
    else:
        self.__tail.next = newNode
        self.__tail = newNode
    self.__size += 1
```

Code
Estrazione di un elemento (dequeue)

Coda di partenza:

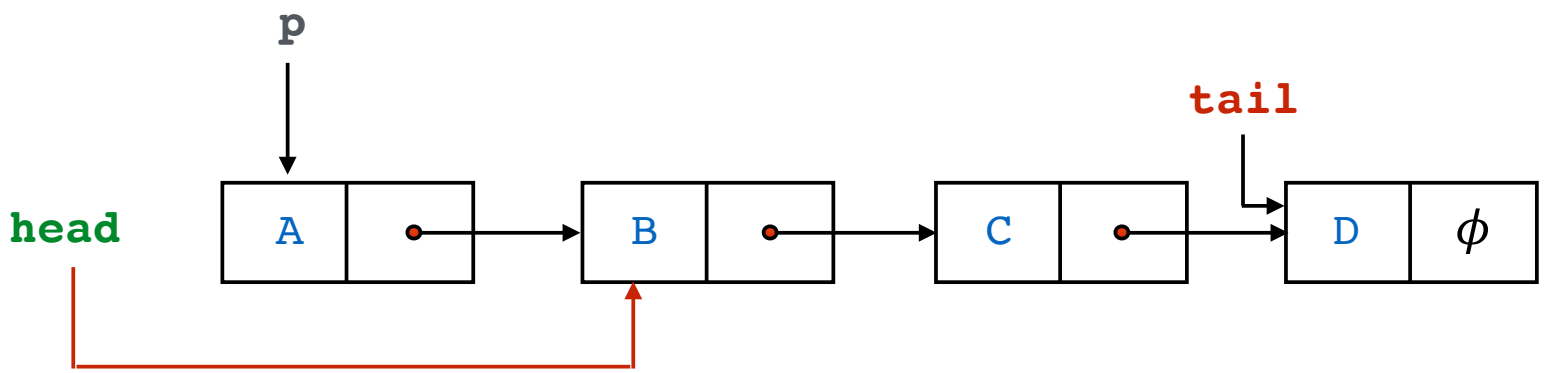


Code
Estrazione di un elemento (dequeue)



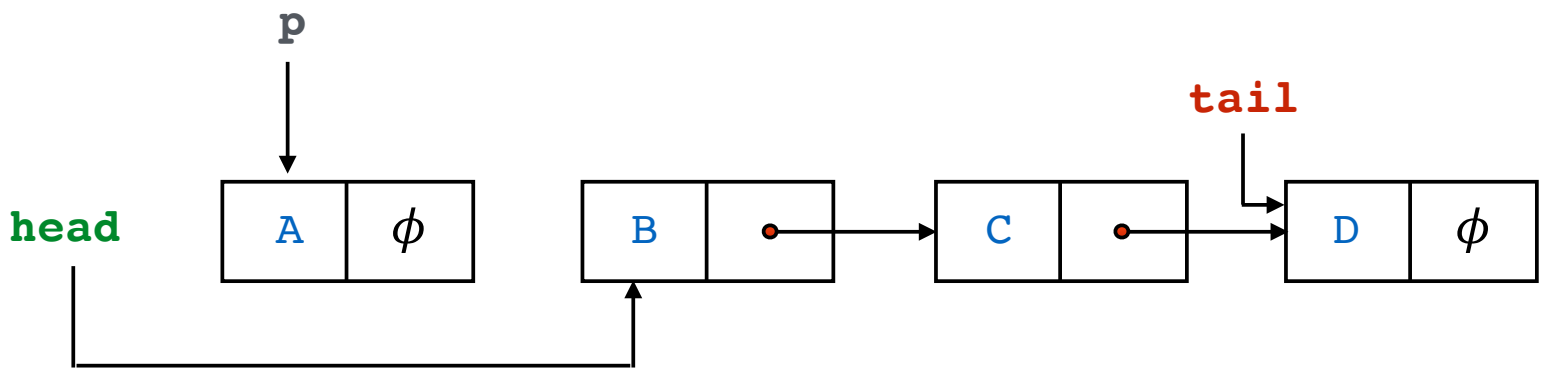
Istruzione: `p = self.__head`

Code
Estrazione di un elemento (dequeue)



Istruzione: `self.__head = self.__head.next`

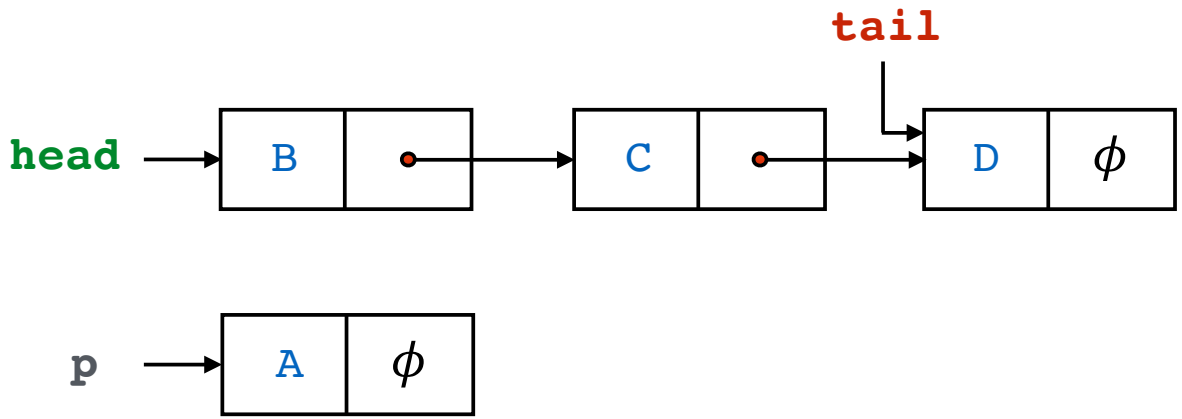
Code
Estrazione di un elemento (dequeue)



Istruzione: `p.next = None`

Code
Estrazione di un elemento (dequeue)

Risultato finale:



Code
Estrazione di un elemento (dequeue)

Codice del metodo **dequeue**:

```
def dequeue(self):
    p = self.__head
    if p == None:
        return None
    self.__head = self.__head.next
    p.next = None
    self.__size -= 1
    return p
```

Riferimenti

Agarwal, B., Baka, B. *Data Structures and Algorithms with Python*, Packt, 2018.

Lubanovic, B. *Introducing Python*, O'Reilly, 2020.

Horstmann, C., Necaise, R.D. *Python for Everyone*, John Wiley & Sons, 2019.

Hu, Y. *Easy learning Data Structures & Algorithms Python*, second edition, 2021.

Aho, A.V., Ullman, J.D. *Fondamenti di Informatica*, Zanichelli, 1994.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. *Introduzione agli algoritmi e strutture dati*, terza edizione, McGraw-Hill, 2010.