

La Programmazione ad Oggetti in Python

Docente: Ambra Demontis

Anno Accademico: 2024 - 2025



University of Cagliari, Italy

Department of Electrical and Electronic Engineering



La Programmazione ad Oggetti in Python

In queste slide vedremo:

- Gli oggetti iteratori
- Gli oggetti iterabili



Supponete di avere una lista e di voler avere una funzione che, ad ogni chiamata, restituisce un elemento della lista (partendo dal primo elemento fino all'ultimo).

Questo si può fare costruendo un oggetto iteratore.

Gli oggetti iteratori sono oggetti particolari il cui scopo è quello di scorrere un oggetto che contiene diversi elementi, restituendo n elementi alla volta.

In Python, la funzione che si utilizza per far si che vengano restituiti i prossimi n elementi è la funzione built-in *next*.

Questa funzione deve essere invocata passandogli come argomento un oggetto iteratore.

Supponiamo di avere una classe di tipo iteratore chiamata CIteratore2Elementi.

Supponiamo inoltre che questa classe si occupi di restituire, due alla volta gli elementi di una lista ricevuta in fase di inizializzazione.

Il comportamento della funzione next sarà quello mostrato nella slide seguente.

```
l = [1,2,3,4,5]
oggetto_iteratore = CIteratore2Elementi(l)
for i in range(10):
 print(next(oggetto_iteratore))
Stamperà:
[1,2]
[3, 4]
Stoplteration
```



Esiste una classe astratta chiamata <u>Iterator</u> che definisce l'interfaccia che gli oggetti di tipo iteratore devono avere.

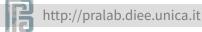
Questa classe astratta può essere importata dal modulo collections.abc

```
from collections.abc import Iterator
```

```
class CProvalteratore(Iterator):
    pass
```

```
oggetto_iteratore = CProvalteratore()
```

TypeError: Can't instantiate abstract class CProvalteratore with abstract methods __next__



Come specificato dalla classe astratta, un oggetto di tipo iteratore deve implementare un metodo __next__.

Questo perchè ciò che fa la funzione built-in next, è richiamare il metodo __next__ implementato dall'oggetto che riceve come argomento.



Il metodo __next__ deve implementare il recupero degli n elementi dall'oggetto.

Deve inoltre generare un'eccezione di tipo *StopIteration* quando vogliamo che non restituisca più altri elementi.

Supponiamo di voler creare un oggetto di tipo iteratore che restituisce, a due a due, gli elementi di una lista e che solleva *StopIteration* quando non ci sono altri due elementi da restituire.



11

from collections.abc import Iterator class CIteratore2Elementi(Iterator): def __init__(self, lista): self._lista = lista self. curr index = 0 def __next__(self): if self._curr_index + 2 <= len(self._lista): elem = self._lista[self._curr_index: self._curr_index+2] self._curr_index = self._curr_index + 2 return elem else: raise StopIteration

Creare una classe iteratore, chiamata CIteratoreBatch che possa essere utilizzato come mostrato nella slide seguente.

Ogni volta che viene chiamata la funzione next sull'oggetto iteratore dovranno quindi essere restituiti n elementi appartenenti alla lista. Se per l'ultimo batch mancheranno degli elementi, dovrà prenderli dall'inizio della lista e poi proseguire da lì.



```
l = [1,2,3,4,5]
oggetto_iteratore = CIteratoreBatch(l,2)
n_iter = 4
for i in range(n_iter):
    print(next(oggetto_iteratore))
```

Dovrà stampare:

[1, 2]

[3, 4]

[5, 1]

[2, 3]

Nb: in pratica rispetto all'esempio dovete gestire il caso in cui l'indice corrente + n è maggiore della lunghezza della lista e non vogliamo fermarci quando non abbiamo più n elementi da restituire

from collections.abc import Iterator

class CIteratoreBatch(Iterator):

def __init__(self, lista, n_elem_batch):
 self._lista = lista
 self._n_elem_batch = n_elem_batch
 self._curr_index = 0



```
def __next__(self):
 end_idx = self._curr_index + self._n_elem_batch
 if end idx <= len(self.lista):
   elem = self._lista[self._curr_index:end_idx]
   self. curr index = end idx
 else:
   n elem mancanti = end idx - len(self.lista)
   elem = self. lista[self. curr index:]
   elem = elem + self._lista[:n_elem_mancanti]
   self. curr index = n elem mancanti
 return elem
```

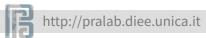


Questo tipo di oggetti viene spesso utilizzato da programmi che implementano algoritmi Machine Learning.

Gli algoritmi di Machine Learning imparano a svolgere dei compiti da degli esempi (insiemi di dati).

Spesso, poichè imaparino ad eseguire un compito correttamente, hanno necessità di vedere tutti quegli esempi più e più volte.

Avere un oggetto iteratore che si occupa di fornire i dati necessari al momento permette di semplificare il resto del codice del programma.



Come abbiamo visto nelle lezioni precedenti, l'istruzione iterativa for può essere utilizzata con istanze di diverse classi che conoscete, es: liste, dizionari, stringhe.

```
Esempio:
lista = [1,2]
for elem in lista:
print(elem)

Stampa:
1
2
```



Può essere utilizzata con un'istanza appartenente ad un oggetto qualsiasi? No! Non saprebbe su quali elementi iterare e come recuperarli...

```
class NuovaLista():
    def __init__(self, lista):
        self.lista = lista

l = [1,2,3]
    oggetto = NuovaLista(l)
for o in oggetto:
        print(o)
```

TypeError: NuovaLista object is not iterable



L'istruzione iterativa for può essere utilizzata su oggetti iterabili.

Gli **oggetti iterabili** sono tutti gli oggetti che **definiscono un metodo chiamato**__iter___ che restituisce un **oggetto iteratore**.

L'oggetto iteratore viene utilizzato dall'istruzione iterativa for per recuperare gli elementi dall'oggetto iterabile.

Supponiamo di voler far si che la classe mostrata prima (*NuovaLista*), composta da una lista, abbia un oggetto iteratore che restituisce, a due a due, gli elementi della lista.

Prima di tutto dobbiamo modificare la classe *NuovaLista*. In particolare dobbiamo aggiungere la definizione del metodo __iter__ che deve restituire un oggetto iterabile.

Dobbiamo poi implementare un oggetto di tipo iteratore.

```
class NuovaLista():

    def __init__(self, lista):
        self._lista = lista

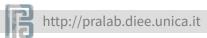
    def __iter__(self):
        return CIteratore2Elementi(self._lista)
```

Per **ottenere un oggetto iteratore da un oggetto iterabile** che implementa il metodo __iter__ si può utilizzare **la funzione** *iter*.

```
l = [1,2,3,4,5]
oggetto = NuovaLista(l)
oggetto_iteratore = iter(oggetto)
print(oggetto_iteratore)
```

Stampa:

<__main__.CIteratore2Elementi object at 0x7f13edaa2dd8>



Anche le strutture dati built-in di python quali liste, dizionari, stringhe come abbiamo detto sono oggetti iterabili, quindi implementano il metodo __iter__.

```
Esempio:
```

```
lista = [1,2]
oggetto_iteratore = iter(lista)
print(oggetto_iteratore)
```

Stampa:

<list_iterator object at 0x7fd4ada66e80>



Una volta estratto l'oggetto iteratore possiamo utilizzarlo per estrarre elementi chiamando la funzione next.

```
lista = [1,2]
oggetto_iteratore = iter(lista)
print(next(oggetto_iteratore))
Stampa:
1
```



Gli oggetti iterabili (anche quelli creati da noi), possiamo scorrerli anche utilizzando l'istruzione iterativa for.

```
l = [1,2,3,4,5]
oggetto = NuovaLista(l)
for o in oggetto:
    print(o)
```

Stampa:

[1, 2]

[3, 4]



Quando può essere utile creare oggetti iterabili?

Ad esempio quando vogliamo poter iterare su oggetti creati da noi.

Supponete di avere una classe CListaVoti che contiene una lista di oggetti di tipo CVoto da voi implementati e supponete di voler poter iterare sugli oggetti di tipo voto utilizzando un ciclo for.

Gli Oggetti Iterabili e Iteratori

Nb: Le classi di oggetti Iterabili e Iteratori sono forniscono un buon esempio di utilizzo di diversi concetti visti a lezione quali:

- -duck typing
- -utilizzo di classi astratte
- -utilizzo delle eccezioni

Esercizio sugli Oggetti Iterabili

Creare una classe *CNuovaStringa* con un attributo *stringa* il cui valore deve essere passato dall'utente al momento dell'inizializzazione dell'oggetto.

Fare in modo che la classe *CNuovaStringa* utilizzi come oggetto iteratore la classe di esempio CIteratore2Elementi.

Utilizzare il ciclo for per verificare se i caratteri vengono effettivamente stampati due a due.

Esercizio sugli Oggetti Iterabili

```
class CNuovaStringa():

    def __init__(self, stringa):
        self._stringa = stringa

    def __iter__(self):
        return CIteratore2Elementi(self._stringa)
```

Esercizio sugli Oggetti Iterabili

```
stringa = "Python"
oggetto = CNuovaStringa(stringa)
for c in oggetto:
 print(c)
Stamperà:
Py
th
on
```

