Programmazione avanzata a.a. 2023-24

A. De Bonis

Introduzione a Python

VII lezione

114

Esercizio su metodi statici e di classe

- Scrivere una classe Impiegato e due sue sottoclassi Tecnico e Amministrativo
- In aggiunta al metodo __init__ , le classe Impiegato puo` avere un solo altro metodo mentre le due sottoclassi possono avere solo __init__
- Scrivere poi un programma che crei un certo numero di istanze delle tre classi e stampa il numero di tecnici, il numero di amministrativi e il numero totale di impiegati (questo potrebbe essere maggiore dalla somma del numero di tecnici e amministrativi perche' è possibile creare impiegati ''generici'' creando direttamente instanze della classe impiegato) .

115

__slots__

- In Python ogni istanza di una classe ha un dizionario (__dict__) che memorizza gli attributi
- Considerando MyClass ed una sua istanza var_c, provate ad eseguire
  - print(MyClass.__dict__)
  - print(var_c.__dict__)
- Spreco di spazio se la classe ha pochi attributi
  - Problema aggravato se si creano tante istanze della classe
- Si può sovrascrivere il comportamento di default definendo __slots__ quando si definisce una classe

__slots__

- A __slots__ si assegna una sequenza di variabili di istanza ed è riservato, in ogni istanza della classe, solo lo spazio sufficiente a memorizzare un valore per ogni variabile
  - __dict__ non sarà più creato
  - non sono più possibili assegnamenti dinamici
    - superabile con __slots__ = . . . , '__dict__'

__slots__

```
>>> class MyNewClass:
...     __slots__='L'
...     def __init__(self,*args):
...         self.L=args
...
>>> var_cn=MyNewClass(1,2,3)
>>> var_cn.L
(1, 2, 3)
>>> var_cn.L=[4,5]
>>> var_cn.L
[4, 5]
```

continua nel riquadro a destra

```
>>> var_cn.X=3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'MyNewClass' object has no attribute 'X'
>>> var_cn.__slots__
'L'
>>> var_cn.__dict__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'MyNewClass' object has no attribute '__dict__'
>>> MyNewClass.__slots__
'L'
```


Riferimenti

- The Python Tutorial
  https://docs.python.org/3/tutorial/
- M.T. Goodrich, R. Tamassia, M.H. Goldwasser
  Data Structures and Algorithms in Python
  Capitolo 2, Object-Oriented Programming

- Studiare anche le sezioni
  - 2.3.3 Multidimensional Vector Class
  - 2.3.5 Range Class
  - 2.4.2 Hierarchy of Numeric Progressions

Distruttore della classe

- Invocato quando si esegue **del** x
- Nella definizione della classe si sovrascrive il metodo __del__
- Viene invocato durante la *garbage collection* quando tutti i riferimenti all'oggetto sono stati eliminati
- **del** x non chiama direttamente x.__del__(), ma decrementa il *reference cout* di x di uno
  - x.__del__() è invocato quando il reference count di x raggiunge il valore 0

120

Distruttore della classe

- __del__ viene invocato quando un'istanza sta per essere distrutta
- Se una classe base ha un metodo __del__ allora il metodo __del__ della classe derivata deve invocare esplicitamente il metodo __del__ della classe base per fare in modo che l'oggetto venga cancellato anche nella parte derivata dalla classe base.

121

Distruttore della classe

```
class Awesome:
    #the init method
    def __init__(self, filename):
        print("Inside the __init__ method.")
        # open file
        self.fobj = open(filename, "w")

    # method
    def writeContent(self, data):
        print("Inside the writeContent method.")
        # write the data
        self.fobj.write(data)

    # the del method
    def __del__(self):
        print("Inside the __del__ method.")
        # close file
        self.fobj.close()

obj = Awesome("helloworld.txt")

obj.writeContent("Hello World")
x=obj
del(obj)
x.writeContent("Hello World")
del(x)
```

```
Inside the __init__ method.
Inside the writeContent method.
Inside the writeContent method.
Inside the __del__ method.
```