Esercizi

Programmazione Avanzata

1

Esercizio 1

- Scrivere una funzione che prende in input una lista L e restituisce una lista di |L|! liste in cui ciascuna lista contiene una diversa permutazione degli elementi della lista input L

2

Esercizio 2

- Scrivere una funzione che prende in input un intero positivo n e restituisce e produce un generatore degli interi 0, 1, 3, 6,10,... . In altre parole, l'i-esimo elemento e` (0+1+2+...+i-1)

3

Esercizio 3

- Definire un decoratore di classe che permette alla classe decorata di contare le sue istanze.

4

Esercizio 4

- Definire un decoratore di funzione che trasforma una funzione che prende in input un numero variabile di numeri in una funzione che prende in input una lista e opera solo sugli elementi della lista di tipo float, int e str convertiti in int.

- la funzione somma non decorata viene invocata in questo modo: somma(3.5, 6, 1.2)
- se usiamo il decoratore, possiamo invocare somma([1.3, 4 ,"6"])

5

Esercizio 5

- Modificare la funzione al punto precedente in modo che la funzione decorata
- operi su qualsiasi elemento della lista che puo` essere convertito in int
- non lanci un'eccezione se un elemento della lista non puo` essere convertito a int
  - cio` puo` non dipendere dal tipo dell'elemento ma dal suo specifico valore, ad esempio "anna"

6

Esercizio 6

- Avete a disposizione la seguente classe

```
class Adapter:
    def __init__(self, obj, adapted_methods):
        self.obj = obj
        self.__dict__.update(adapted_methods)
    def __str__(self):
        return str(self.obj)
```

e le classi Lavoratore, Commesso, Cuoco, Musicista (illustrate nella prossima slide). La classe Lavoratore e` stata implementata da voi mentre le restanti sono in una libreria esterna il cui codice sorgente non puo` essere modificato.

7

Esercizio 6

```
class Lavoratore:
    def __init__(self, nome):
        self.nome = nome
    def __str__(self):
        return "il lavoratore {}".format(self.nome)
    def lavora(self,lavoro): return "svolge il seguente {}".format(lavoro)
```

```
class Commesso:
    def __init__(self, nome):
        self.nome = nome
    def __str__(self): return "il commesso  {}".format(self.nome)
    def cucina(self,merce): return "vende {}".format(merce)
```

```
class Musicista:
    def __init__(self, nome):
        self.nome = nome
    def __str__(self):
        return "il musicista  {}".format(self.nome)
    def suona(self,tipoMusica): return "suona
{}".format(tipoMusica)
```

```
class Cuoco:
    def __init__(self, nome):
        self.nome = nome
    def __str__(self): return "il cuoco  {}".format(self.nome)
    def cucina(self,pietanza): return "cucina {}".format(pietanza)
```

8

Esercizio 6

- Scrivere un programma che stampa le seguenti stringhe utilizzando solo i metodi di Adapter e Lavoratore:
- Il commesso Paolo vende abiti
- Il musicista Veronica suona musica pop
- Il cuoco Antonio cucina una lasagna

9

Esercizio 7

- Scrivere una classe di base ClsBase in cui c'e` un metodo addAttr che
  - prende in input **due** argomenti: una stringa s e un valore v,
  - controlla se la classe ha l'attributo di nome s e se tale attributo non e` presente allora aggiunge alla classe l'attributo s con valore v; in caso contrario non fa niente.
  - Il metodo deve funzionare anche per le eventuali sottoclassi di ClsBase

10

Esercizio 8

- Scrivere una classe che contiene un metodo che restituisce il numero di invocazioni degli altri metodi della classe. Il codice dei suddetti metodi non deve essere modificato.

11

Esercizio 9

- Scrivere un decorator factory che genera un decoratore di classe che dota la classe di un metodo che restituisce il numero di invocazioni del metodo passato come parametro al decorator factory.

12

Esercizio 10

Si considerino le classi Cane e Persona fornite nel file **modulo.py** presente sul sito. Scrivere la classe Casa con due cani e una persona (padrona del cane). La classe Casa fa uso di un mediatore per fare in modo che

• quando almeno uno dei due cani abbaia allora viene settata a True un flag di allerta (variabile self.allerta nella bozza di __init__ è fornita nel file di test **esercizioMediator.py** presente sul sito).

• quando il padrone torna a casa, se il flag allerta e` True, verifica per ciascun cane se tra l'ora in cui e` tornato a casa e l'ora in cui il cane ha mangiato per l'ultima volta sono trascorse piu` di 4 ore e in questo caso da` da mangiare al cane. Se nessuno dei due cani ha abbaiato tra il momento in cui il padrone e` uscito e quello in cui ha fatto ritorno (il flag e` False) allora il padrone al suo ritorno non fa niente.

NB: puo` essere che il cane che abbaia non sia quello che ha fame o che ne abbai uno solo ma che entrambi abbiano fame, o ancora che almeno uno dei cani abbai ma nessuno dei due abbia fame.

Suggerimento: Per ciascuno dei due punti creare un callable: uno dei due deve essere associato ad entrambi i cani e l'altro deve essere associato al padrone.

La differenza in ore tra due orari ora1 e ora2 si calcola cosi` : (ora1-ora2).total_seconds()/60/60 .

Esercizio 11

1. Scrivere una funzione **ricorsiva** che prende in input una lista e un elemento x. La funzione restituisce True non appena trova x nella lista. Se x non e` nella lista, la funzione restituisce False
2. Scrivere la funzione **ricorsiva** myDeepCopy che prende in input una lista che potrebbe contenere al suo interno elementi di tipo lista che a loro volta potrebbero contenere elementi di tipo lista, e cosi` via. La funzione restituisce la deep copy della lista (no, non si puo` usare copy.deepcopy).

   Esempi di input: [] , [1, [2, 3]] , [[],[1, [2, 3]]] , [[[5, 4]][[2], 3],5]

Scrivere anche una funzione main che testi il corretto funzionamento di myDeepCopy

Esercizio 12

3. Scrivere la classe MyDictionary che implementa gli operatori di dict riportati di seguito. MyDictionary deve avere **solo** una variabile di istanza e questa deve essere di tipo lista. Per rappresentare le coppie, dovete usare la classe MyPair che ha due variabili di istanza (key e value) e i metodi getKey, getValue, setKey, setValue .

d[key]	value associated with given key
d[key] = value	set (or reset) the value associated with given key
del d[key]	remove key and its associated value from dictionary
key **in** d	containment check
key **not in** d	non-containment check
d1 == d2	d1 is equivalent to d2
d1 != d2	d1 is not equivalent to d2

15