Estensione di `RUB`: una classe rubrica

Barbara guidi, Susanna Pelagatti

Secondo assegnamento in itinere 622AA
AA 2021-22

Indice

1 Introduzione

Il primo modulo (9 crediti) del corso di Programmazione e Analisi Dati (622AA) prevede lo svolgimento di due assegnamenti in itinere che esonerano dallo svolgimento del progetto finale. Questo documento descrive il secondo assegnamento che prevede l'estensione della classe Python sviluppata durante il primo assegnamento.

Il software viene sviluppato e documentato utilizzando gli strumenti, le tecniche e le convenzioni presentati durante il corso.

1.1 Materiale in linea

Tutto il materiale relativo al corso può essere reperito su TEAMS (registrazione delle lezioni) e sul Moodle ufficiale del corso.

Eventuali chiarimenti possono essere richiesti alle docenti per posta elettronica.

1.2 Struttura degli assegnamenti, bonus, tempi di consegna e prova orale

I due assegnamenti possono essere svolti individualmente o in gruppi di 2 studenti e possono essere consegnati entro la data e l'ora dell'appello di Febbraio 2022. Agli studenti che consegnano una realizzazione sufficiente entro 8 giorni dalla pubblicazione su Moodle vengono assegnati 2 punti di Bonus nella valutazione complessiva di ciascun assegnamento. La valutazione complessiva viene

data dalla media delle valutazioni dei due assegnamenti e costituisce la base per la prova orale. La prova orale tenderà a stabilire se lo studente è realmente l'autore degli assegnamenti consegnati e verterà su tutto il programma del corso e su tutto quanto usato negli assegnamenti anche se non fa parte del programma del corso. Il voto dell'orale (da 0 a 30L) fa media con la valutazione degli assegnamenti per delineare il voto finale. In particolare, l'orale comprenderà

- una discussione delle scelte implementative
- l'impostazione e la scrittura di semplici programmi Python (sequenziali e concorrenti)
- domande su tutto il programma presentato durante il corso.

1.3 Consegna degli assegnamenti

La consegna degli assegnamenti avviene *esclusivamente* per posta elettronica secondo le istruzioni presenti nel `README` di ciascun assegnamento.

1.4 Valutazione dell'assegnamento

All'assegnamento viene assegnata una fascia di valutazione da 0 a 30 che tiene conto dei seguenti fattori:

- motivazioni, originalità ed economicità delle scelte implementative
- strutturazione del codice (fattorizzazione del codice in funzioni, uso di strutture dati adeguate etc)
- efficienza e robustezza (numero di operazioni eseguite, fallimenti in caso di input inadeguati etc)
- aderenza alle specifiche
- qualità del codice Python e dei commenti

Tutti gli assegnamenti verranno confrontati automaticamente per verificare situazioni di plagio. Nel caso di elaborati uguali verranno presi provvedimenti per tutti i gruppi coinvolti.

2 Secondo assegnamento

L'assegnamento prevede l'estensione della classe `Rubrica`, in modo da creare una rubrica Thread-safe. L'accesso alla rubrica deve essere regolato da opportuni meccanismi di sincronizzazione per evitare situazione di inconsistenza dei dati.

E' necessario individuare i metodi che possono portare ad uno stato di inconsistenza dei dati, tra quelli proposti durante il primo assegnamento e riportati di seguito, ed applicare uno dei meccanismi di sincronizzazione visto a lezione (Lock, RLock, o Conditions), per poter sincronizzare l'accesso alla struttura dati Rubrica.

I metodi del primo assegnamento sono:

```
class Rubrica:
""" Costruttore: crea una rubrica vuota rappresentata come
    dizionario di contatti vuoto """
def __init__(self):
    """ crea una nuova rubrica vuota """
    self.rub = {}
def  __str__(self):
    """ Serializza una rubrica attraverso una stringa
    con opportuna codifica (a scelta dello studente) """
def __eq__(self,r):
    """stabilisce se due rubriche contengono
    esattamente le stesse chiavi con gli stessi dati"""
def __add__(self,r):
    """crea una nuova rubrica unendone due (elimina i duplicati)
    e la restituisce come risultato --
    se ci sono contatti con la stessa chiave nelle due rubriche
    ne riporta uno solo """
def load(self, nomefile):
    """ carica da file una rubrica eliminando il
    precedente contenuto di self """
def inserisci(self, nome, cognome, dati):
    """ inserisce un nuovo contatto con chiave (nome,cognome)
    restituisce "True" se l'inserimento Ã¨ andato a buon fine e
    "False" altrimenti (es chiave replicata) -- case insensitive """
def modifica(self, nome, cognome, newdati):
    """ modifica i dati relativi al contatto con chiave (nome,cognome)
    sostituendole con "newdati" -- restituisce "True" se la modifica
    Ã¨ stata effettuata e "False" altrimenti
    (es: la chiave non e' presente )"""
def cancella(self, nome, cognome):
    """ il contatto con chiave (nome,cognome) esiste lo elimina
    insieme ai dati relativi e restituisce True -- altrimenti
    restituisce False """
def cerca(self, nome, cognome):
    """ restitusce i dati del contatto se la chiave e' presente
    nella rubrica e "None" altrimenti -- case insensitive """
def store(self, nomefile):
    """ salva su file il contenuto della rubrica secondo
    un opportuno formato (a scelta dello studente)"""
    # il formato da me scelto
    # prevede un contatto per linea
    # nome:cognome:telefono\n
def ordina(self,crescente=True):
    """ serializza su stringa il contenuto della rubrica come in
        Nannipieri Felice 32255599\n
        Neri Paolo 347555776\n
        Rossi Mario 3478999\n
        Rossi Mario Romualdo 3475599\n
        Tazzini Tazzetti Gianna 33368999\n
        le chiavi ordinate lessicograficamente per Cognome -- Nome
```

```
        in modo crescente (True) o decrescente (False)
        Fra nome, cognome e telefono seve essere presente
        ESATTAMENTE uno spazio
        Restituisce la stringa prodotta """
```

Creare due nuovi metodi nella classe Rubrica:

- *suggerisci.*Il metodo suggerisci viene invocato da un thread per effettuare un suggerimento di un contatto nella rubrica. Prende come parametro il nome ed il cognome del contatto e lo inserisce in una coda (di lunghezza massima 3). Non si puo' inserire un elemento nella coda se la coda e' piena.
- *suggerimento.* Il metodo suggerimento viene invocato da un thread per ottenere un suggerimento recuperato dalla rubrica. Il thread legge gli elementi presenti in una coda di lunghezza 1. Se la coda é vuota, attende l'inserimento di un elemento, altrimenti prende il primo elemento della coda e lo stampa.

```
def suggerisci(self, nome, cognome):
"""Il metodo suggerisci viene invocato da un thread per
effettuare un suggerimento di un contatto nella rubrica.
Prende come parametro il nome ed il cognome del contatto e
lo inserisce in una coda (di lunghezza massima 3).
Non si puo' inserire un elemento nella coda se la coda e' piena.
"""
def suggerimento(self):
"""Il metodo suggerimento viene invocato da un thread per
ottenere un suggerimento recuperato dalla rubrica.
Il thread legge gli elementi presenti in una coda
di lunghezza 3. Se la coda Ã¨ vuota, attende l'inserimento di
un elemento, altrimenti prende il primo elemento della
coda e lo stampa."""
```

2.1 Istruzioni

Unitamente a questo file vengono forniti due files: la classe Produttore e la classe Consumatore. Le due classi estendono la classe Thread ed utilizzano la rubrica come un oggetto comune. Entrambi i thread possono chiamare tutti i metodi dell'oggetto Rubrica, tranne i due nuovi metodi *suggerisci* e *suggerimento*. Il metodo suggerisci può essere invocato solo dal Produttore per suggerire un nominativo presente in rubrica, e il consumatore può invocare il metodo suggerimento, per recuperare i suggerimenti forniti dal Produttore.

Le classi sono già implementate e definiscono il comportamento dei due thread e le azioni che andranno a svolgere. Utilizzare le due classi fornite per testare l'accesso a Rubrica.

Implementare la classe main.py. La classe deve fornire una interfaccia grafica che permetta di testare il comportamento di Produttore e Consumatore. In particolare, l'interfaccia deve fornire almeno due funzionalità:

- **Simple Test.** Avviare un Thread Produttore ed un Thread Consumatore.

- **Test Multithread.** Tramite l'interfaccia deve essere possibile inserire il numero di thread (Produttori e Consumatori) da avviare. Si consideri N il numero preso in input da interfaccia. Si devono avviare N Produttori ed N Consumatori.

Possono essere realizzate funzionalità in più rispetto a quelle richieste.

Per la costruzione dell'interfaccia utilizzare il modulo tkinter.

Per la gestione delle stampe, considerato il problema della print visto a lezione, Ã¨ necessario utilizzare il modulo logging.

Consigliamo di leggere attentamente le istruzioni per capire come strutturare la classe prima di iniziare a progettare e implementare. Ricordiamo l'importanza di analizzare i vari casi prima di iniziare a scrivere codice e di effettuare test incrementali sul codice durante lo sviluppo dei vari metodi.

Le parti opzionali devono essere corredate da test appropriati e documentate da commenti chiari o (in caso sia necessario) da un breve documento descrittivo che puo' essere consegnato insieme al codice e che spiega le motivazioni e la struttura di quanto realizzato.