Laboratorio di Programmazione

Compito d'esame per l'appello del 9/2/2021

Una *time series* (univariata) è una serie di coppie di punti dove il primo elemento della coppia è un istante di tempo, anche detto *timestamp,* ed il secondo è il valore di una qualche quantità relativa a quell'istante, come ad esempio la temperatura.
Il timestamp può essere rappresentato in vari formati, uno dei più comuni in informatica è l'*epoch*, che sta a rappresentare il numero di secondi passati dalla mezzanotte (00:00) del primo Gennaio 1970 sulla timezone UTC (ovvero il meridiano fondamentale di Greenwich, senza cambi di ora legale). All'una di notte del primo Gennaio 1970 il timestamp epoch vale quindi "3600" secondi, alle due vale "7200" secondi, e così via.
Al momento della scrittura di questo testo, il 2 Febbraio 2021 alle 16:20 in Italia, ovvero alle 15:20 UTC, l'epoch vale "1612279200" secondi. Tra un minuto questo epoch sarà incrementato di 60 secondi (e varrà "1612279260"). E domani alla stessa ora sarà incrementato di 86400 secondi (60 secondi volte 60 minuti volte 24 ore) e varrà quindi "1612365600". Questo sito fornisce un rapido modo di convertire una data e ora in epoch e viceversa: epochconverter.com.
Il file **data.csv** contiene la time series della temperatura dell'interno di un'abitazione, registrata una volta all'ora per un mese (Marzo 2019), e si presenta così:

```
epoch,temperature
1551398400,21.50
1551402000,21.40
1551405600,21.30
...
```

overo, messa sotto forma di tabella per comodità:

epoch	temperature
1551398400	21.50
1551402000	21.40
1551405600	21.30
...	...

Vogliamo leggere questo tipo di dati e calcolare le statistiche giornaliere di temperatura minima, massima e media. Per "statistiche giornaliere" si intende che si devono processare tutte le misurazioni orarie appartenenti ad un dato giorno e calcolarne appunto il valore minimo, massimo e medio, ripetendo il procedimento per ogni giorno della time series.

Informazioni sullo svolgimento

Assumiamo che il file della serie temporale da leggere e su cui vogliamo calcolare le statistiche giornaliere sia ***sempre di un mese esatto di dati***. Questo ci permette di non dover avere a che fare con i calendari per capire di che mese si tratta, il che è una cosa che al momento non ci interessa. Non dovete quindi preoccuparvi di controllare che i dati letti dal file CSV siano tutti appartenenti allo stesso mese: lo sono da specifiche.
Occhio invece che non si parla di uno specifico mese, quindi potete aspettarvi misurazioni orarie appartenenti ad un mese di 31 giorni (Marzo, come nell'esempio) ma anche da 30, da 29 o da 28 giorni. La chiave dello svolgimento dell'esame sta nel capire, dato il timestamp epoch delle misurazioni orarie, se quest'ultime appartengono allo stesso giorno (nota bene: non a ***quale*** giorno, ma solo se allo ***stesso*** giorno). Su questo può aiutarvi il suggerimento sull'utilizzo dell'operazione modulo nella sezione "Qualche informazione in più" in fondo.

Possono esserci dei dati mancanti, quindi attenzione a come calcolate l'inizio e la fine dei vari giorni e ai conti che fate. Tuttavia, se mancano dati, questi non mancano mai per più di un giorno (ovvero, ***per ogni giorno è presente almeno una misurazione di temperatura***).
I giorni vengono considerati tutti di lunghezza costante di 86400 secondi siccome siamo sulla timezone UTC (dove non si considera mai il passaggio all'ora legale, che invece nella realtà è parecchio fastidioso).
Alla luce di tutto questo, create la classe `CSVTimeSeriesFile`, modificando o estendendo la classe `CSVFile` vista a lezione (oppure scrivendola da zero). La classe deve essere istanziata sul nome del file tramite la variabile `name` e deve avere un metodo `get_data()` che torni una lista di liste, dove il primo elemento delle liste annidate è l'epoch ed il secondo la temperatura.
Questa classe si dovrà quindi poter usare così:

```
time_series_file = CSVTimeSeriesFile(name='data.csv')
time_series = time_series_file.get_data()
```

...ed il contenuto della variabile `time_series` tornato dal metodo `get_data()` dovrà essere così strutturato (come lista di liste):

```
[
  [1551398400, 21.50],
  [1551402000, 21.40],
  [1551405600, 21.30],
  ...
]
```

Per calcolare le statistiche giornaliere dovete invece creare una funzione a sé stante (cioè posizionata non nella classe `CSVTimeSeriesFile` ma direttamente nel corpo principale del programma), di nome `daily_stats`, che verrà usata così:

```
daily_stats(time_series)
```

..e che dovrà ritornare (tramite un `return`) in output sempre una lista di liste, dove però ogni lista annidata rappresenta la statistica giornaliera di un dato giorno, ovvero la tripletta di temperatura minima, massima e media:

```
[
  [valore_min_giorno_1, valore_max_giorno_1, valore_medio_giorno_1],
  [valore_min_giorno_2, valore_max_giorno_2, valore_medio_giorno_2],
  ...
]
```

Il file in cui scrivere il vostro codice deve chiamarsi "esame.py" e le eccezioni da alzare in caso di input non corretti o casi limite devono essere istanze di una specifica classe `ExamException`, che dovete definire nel codice come segue, senza modifica alcuna (copia-incollate le due righe):

```
class ExamException(Exception):
    pass
```

...e che poi userete come una normale eccezione, ad esempio:

```
raise ExamException('Errore, lista valori vuota')
```

Qualche informazione in più sulle specifiche e qualche e suggerimento:

- I timestamp epoch che leggete da file sono di tipo intero. Se per caso dovessero esserci dei timestamp epoch floating point, vanno convertiti silenziosamente ad interi (tramite cast diretto con `int()`, non tramite arrotondamento) e tutto deve procedere comunque.

- I valori di temperatura che leggete dal file CSV sono da aspettarsi di tipo numerico (intero o floating point), un valore di temperatura non numerico, oppure vuoto o nullo non deve essere accettato, ma tutto deve procedere comunque senza alzare eccezioni.
- La serie temporale nel file CSV è da considerare sempre ordinata, se per caso ci dovesse essere un timestamp fuori ordine va alzata un'eccezione senza cercare di riordinare la serie. Stesso discorso se c'è un timestamp duplicato: si alza un'eccezione.
- Il file CSV può contenere letteralmente di tutto. Da linee incomplete a pezzi di testo che non c'entrano niente, e ogni errore *salvo quello di un timestamp fuori ordine o duplicato* va ignorato (ovvero, ignoro la riga contenente l'errore e vado a quella dopo).

- Se leggete correttamente una serie temporale dal file CSV, questa (come già accennato precedentemente) è assicurato che sia di un mese esatto di dati e con almeno una misurazione di temperatura per giorno.

- La classe `CSVTimeSeriesFile` controlla l'esistenza del file solo quando viene chiamato il metodo `get_data()` e, nel caso il file non esista o non sia leggibile, alza un'eccezione.
- Suggerimento: per trovare l'inizio di un giorno dato un timestamp epoch, potete usare l'operazione modulo che calcola il resto di una divisione, e sottrarlo al timestamp stesso, in questo modo: `day_start_epoch = epoch - (epoch % 86400)`
- Attenzione alle condizioni nel controllo dell'appartenenza di un timestamp epoch ad un determinato giorno: la mezzanotte appartiene sempre al giorno dopo, anche sull'orologio che avete sul polso / cellulare / comodino!

Informazioni sulla consegna

Quando avete sviluppato quanto richiesto nel file esame.py, commitatelo in un vostro repository su GitHub (o Bitbucket o GitLab). La consegna va effettuata il giorno dell'appello condividendo in chat su Teams o via email il link al repository e l'hash da valutare. Riassunto sulla modalità di esame: https://sarusso.github.io/ProgrammingLab/#esami
Attenzione: se il file non si chiama "esame.py", se le eccezioni alzate in caso di errori non sono di tipo "`ExamException`" o se le classi ed i metodi non si chiamano come indicato da specifiche, l'esame non potrà essere valutato!
Se non vi ricordate bene come si usa Git, potete fare riferimento al tutorial dei tutor che è disponibile qui: https://github.com/drpOpZ/proglab2021-tutors/blob/master/git_quickstart.md.