Logging in Python

Introduzione

- Un log è una lista di eventi generalmente considerata di interesse per almeno una delle seguenti figure
  - Sviluppatore software: l'applicazione ha dei bachi?
  - Amministratore di rete/sistema: l'applicazione sta funzionando? Se sì, con che livello di prestazione?
- Un log è assimilabile ad una lista di record testuali
  - Ogni record rappresenta un evento
  - Formato CSV (Comma Separated Values), dove la "Comma" in realtà è spesso uno spazio bianco

Eventi

- Un evento è identificato attraverso la risposta alle seguenti domande
  - Cosa è successo?
  - Quando è successo?
  - Dove è successo?
  - Quanto è importante?
- I primi tre elementi (cosa, quando, dove) sono oggettivi
- Il quarto elemento è soggettivo
  - Cosa è degno di essere registrato?
  - È possibile differenziare l'importanza dell'evento?

Logger

- Il logger è il componente software adibito alle mansioni di logging
  - Nei linguaggi dinamici moderni, è un modulo/package
- Compiti di un logger
  - Permettere in maniera semplice la memorizzazione (record) di eventi (cosa, quando, dove, quanto importante)
  - Permettere ad altri utenti di leggere il contenuto del log (anche da remoto, se necessario)
  - Abilitare in maniera semplice una configurazione alternativa della modalità di logging

Logger

- Come implementa un logger i requisiti ora visti?
- Cosa è successo
  - Il logger riceve una stringa di formato con argomenti opzionali e costruisce il record dell'evento
- Quando è successo
  - Il logger attacca un marcatore temporale (time stamp) al record; il marcatore riflette l'istante in cui il record è stato inserito nel log

Logger

- Come implementa un logger i requisiti ora visti?
- Dove è successo
  - Il logger può mantenere diverse liste di eventi (dette anche canali di log), una per ogni area
- Quanto è importante
  - L'utente associa priorità intere (livelli di debug) all'evento

Il modulo logging: apertura di un canale

- Python mette a disposizione il modulo logging per l'implementazione di molteplici canali di logging

```
import logging
```

- Un canale di log può essere aperto attraverso il metodo getLogger()

```
csvLogger = logging.getLogger("input.csv")
```

- Sono disponibili diversi canali
  - File di testo (esempio più comune)
  - Socket di rete

Il modulo logging: i livelli di log

- Sono definiti i seguenti livelli di debug
  - DEBUG: informazioni dettagliate sulla applicazione, utili per chi deve effettuare il debugging della stessa
  - INFO: messaggi informativi che indicano il corretto procedere delle operazioni
  - WARNING: sussiste una anomalia; l'applicazione sta funzionando bene, ma qualcosa potrebbe andare storto in futuro
  - ERROR: l'applicazione non è stata in grado di effettuare una operazione, ma può continuare ad operare
  - CRITICAL: l'applicazione è sul punto di abortire definitivamente le proprie operazioni

Il modulo logging: i livelli di log

- Tali livelli possono essere indicati attraverso i corrispettivi metodi
- DEBUG

```
csvLogger.debug("Trying to read file '%s'" % filename)
```

- WARNING

```
csvLogger.warning("File '%s' contains no data",
filename)
```

- ERROR

```
csvLogger.error("File '%s': unexpected end of file at line
%d, offset %d", filename, lineno, offset)
```

- CRITICAL

```
csvLogger.critical("File '%s': too large, not enough
memory, amount used = %d", filename, memused)
```

Il modulo logging: logging delle eccezioni

- È possibile effettuare il logging di una eccezione attraverso il metodo exception()

```
logger.exception("Error reading file '%s' at offset %d",
filename, offset)
```

- In alternativa, si possono usare i metodi visti precedentemente ed impostare ad 1 il parametro con nome exc_info

```
logger.error("Error reading file '%s' at offset %d",
filename, offset, exc_info=1)
```

Il modulo logging: impostazione del livello

- È possibile impostare il livello di default di logging tramite il metodo setLevel()

```
logger.setLevel(INFO)
```

- È possibile abilitare l'elaborazione di un messaggio di logging in maniera dinamica, solo se il livello di logging è effettivamente impostato

```
if logger.isEnabledFor(logging.DEBUG):
    logger.debug("Message with %s, %s", \
    expensive_func1(), expensive_func2())
```

Il modulo logging: configurazione

- La configurazione avviene attraverso il metodo basicConfig()

```
logging.basicConfig(level=logging_level,
    filename=options.logging_file, \
    format='%(asctime)s %(levelname)s: %(message)s', \
    datefmt='%Y-%m-%d %H:%M:%S')
```

- Argomenti
  - level: livello di logging
  - filename: nome del file
  - format: stringa di formato
    - asctime:data in formato stringa
    - levelname: nome del livello di debug
    - message: record
    - datefmt: formato della data

Un esempio

- ESEMPI:
python-logger.py

- **Si esegua il seguente comando**

./python-logger.py –logging-level=debug
--logging-file=debug.log

Configurazione in Python

Il modulo ConfigParser

- Python mette a disposizione il modulo ConfigParser per la lettura e scrittura di file di configurazione nel formato .ini

```
import ConfigParser
```

- Si crea un oggetto di tipo ConfigParser che rappresenta una configurazione

```
Config = ConfigParser.ConfigParser()
```

Il modulo ConfigParser

- Si importa la configurazione da file

```
Config.read("config.ini")
```

- Si recupera la configurazione in un dizionario attraverso due metodi
  - sections(): ritorna le sezioni di primo livello
  - options(section): ritorna le voci di configurazione di una sezione

```
options = Config.options(section)
for option in options:
    dict1[option] = Config.get(section, option)
    ...
```