# Introduzione

Questo libro si rivolge ai programmatori in Python che desiderano ampliare e approfondire la loro conoscenza del linguaggio in modo da poter migliorare la qualità, l'affidabilità, la velocità, la facilità di manutenzione e di utilizzo dei loro programmi. Presenta numerosi esempi pratici e idee per migliorare la programmazione in Python.
Il libro è centrato su quattro temi chiave: i design pattern per scrivere codice in modo elegante ed efficace, la concorrenza e Cython (Python compilato) per aumentare la velocità di elaborazione, l'elaborazione di rete ad alto livello e la grafica.
Il volume *Design Patterns: Elements of Reusable Object-Oriented Software* (cfr. la bibliografia a fine libro per i dettagli) è stato pubblicato già nel 1995 e tuttavia continua a esercitare una forte influenza sulle pratiche di programmazione orientata agli oggetti.
Questo libro esamina tutti i design pattern nel contesto di Python, fornendo esempi per quelli ritenuti più utili, e spiegando perché alcuni sono irrilevanti per i programmatori. Questi pattern sono trattati nei Capitoli 1, 2 e 3.
Il GIL (*Global Interpreter Lock*) di Python evita che il codice sia eseguito su più di un core del processore alla volta.

> **NOTA**
>
> Questa limitazione si applica a CPython, l'implementazione di riferimento che la maggior parte dei programmatori in Python utilizza. Alcune implementazioni di Python non hanno questo limite, in particolare Jython – Python implementato in Java.

Questo ha fatto nascere il mito secondo cui Python non sarebbe in grado di gestire i thread o di trarre vantaggio dall'hardware multi-core. Per quanto riguarda l'elaborazione CPU-bound, la concorrenza può essere gestita utilizzando il modulo `multiprocessing`, che non è limitato dal GIL e può sfruttare al massimo tutti i core disponibili. In questo modo è facile ottenere i miglioramenti attesi nella velocità (circa proporzionali al numero di core). Anche per l'elaborazione I/O-bound possiamo utilizzare il modulo `multiprocessing`, oppure il modulo `threading`, o il modulo `concurrent.futures`. Se utilizziamo il threading per la concorrenza I/O-bound, solitamente la latenza di rete prevale rispetto al carico aggiuntivo comportato da GIL, che perciò non costituisce un problema significativo nella pratica.

Sfortunatamente gli approcci di basso e medio livello alla concorrenza tendono a generare errori (in qualsiasi linguaggio). Possiamo evitare tali problemi scegliendo di non utilizzare lock espliciti e sfruttando i moduli `queue` e `multiprocessing` di Pyhton, oppure il modulo `concurrent.futures`. Vedremo come ottenere significativi miglioramenti delle prestazioni utilizzando la concorrenza di alto livello nel Capitolo 4.

A volte i programmatori utilizzano C, C++ o altri linguaggi compilati a causa di un altro mito: che Python sarebbe lento. Benché Python sia in generale più lento dei linguaggi compilati, su hardware moderno in realtà è spesso veloce quanto basta per la maggior parte delle applicazioni. E nei casi in cui Python non è veloce a sufficienza, possiamo comunque godere dei benefici di programmare in questo linguaggio e nello stesso tempo di ottenere un'esecuzione più veloce del codice.

Per velocizzare i programmi a lunga esecuzione possiamo utilizzare l'interprete Python PyPy (`http://pypy.org`), che dispone di un compilatore just-in-time in grado di fornire notevoli miglioramenti nelle prestazioni. Un altro modo per migliorare le performance è quello di utilizzare codice che sia eseguito velocemente quasi come il codice C compilato; nel caso dell'elaborazione CPU-bound si possono ottenere incrementi di velocità dell'ordine delle 100 volte. Il modo più facile per ottenere una velocità simile a quella del C è quello di utilizzare moduli Python che siano già scritti in C: per esempio il modulo `array` della libreria standard o il modulo esterno `numpy` per elaborare gli array con incredibile velocità e grande efficienza per quanto riguarda la memoria (anche array multidimensionali con `numpy`). Un altro modo è quello di eseguire una profilatura utilizzando il modulo `cProfile` della libreria standard per scoprire dove si trovano i colli di bottiglia, e poi scrivere codice per cui la velocità è un aspetto critico utilizzando Cython – che in sostanza fornisce una sintassi Python migliorata che viene compilata in C puro per la massima velocità in fase di runtime.

Naturalmente, a volte le funzionalità che ci servono sono già disponibili in una libreria in C o C++, o di un altro linguaggio che utilizza la convenzione di chiamata del C. Nella maggior parte dei casi esiste un modulo Python esterno che fornisce l'accesso alla libreria che ci serve – questi moduli si possono trovare nel Python Package Index (PyPI; `http://pypi.python.org`). Tuttavia, per i rari casi in cui un tale modulo non sia disponibile, si può usare il modulo `ctypes` della libreria standard per accedere alla funzionalità di librerie in C, come può fare il pacchetto esterno Cython. L'uso di librerie C preesistenti può ridurre notevolmente i tempi di sviluppo, e solitamente consente di ottenere una notevole velocità di elaborazione. Cython e `ctypes` sono entrambi trattati nel Capitolo 5.

La libreria standard di Python fornisce una varietà di moduli per l'elaborazione di rete, dal modulo di basso livello `socket` a quello di livello intermedio `socketserver`, fino a quello di alto livello `xmlrpclib`. Benché l'elaborazione di rete di livello basso e intermedio abbia senso quando si effettua il porting del codice da un altro linguaggio, partendo subito in Python possiamo spesso evitare i dettagli di basso livello e concentrarsi su ciò che le nostre applicazioni di rete devono fare utilizzando moduli di alto livello. Nel Capitolo 6 vedremo come fare ciò utilizzando il modulo `xmlrpclib` della libreria standard e il modulo esterno `RPyC`, potente e facile da usare.

Quasi tutti i programmi devono fornire un'interfaccia utente di qualche tipo affinché sia possibile indicare che cosa fare. I programmi in Python possono essere scritti in modo da supportare interfacce utente a riga di comando, con il modulo `argparse`, e interfacce utente tipo terminale (per esempio su Unix usando il pacchetto esterno `urwid`; `http://`

excess.org/urwid). Esistono anche molti ottimi framework web, dal semplice e leggero bottle (http://bottlepy.org) ai colossi come Django (http://www.djangoproject.com) e Pyramid (http://www.pylonsproject.org), che possono tutti essere utilizzati per fornire alle applicazioni un'interfaccia web. E naturalmente Python può essere usato per creare applicazioni GUI (*Graphical User Interface*).

Spesso si sente dire che le applicazioni GUI sarebbero destinate a scomparire in favore delle applicazioni web, ma non è ancora successo. In effetti le persone sembrano preferire le applicazioni GUI alle applicazioni web. Per esempio, quando è iniziato il boom degli smartphone all'inizio del ventunesimo secolo, gli utenti preferivano sempre utilizzare una "app" appositamente creata, anziché un browser e una pagina web per attività che svolgevano regolarmente. Esistono molti modi per programmare GUI in Python usando pacchetti esterni. Nel Capitolo 7 vedremo come creare applicazioni GUI moderne usando Tkinter, fornito con la libreria standard di Python.

La maggior parte dei computer moderni, tra cui i notebook e anche gli smartphone dispone di potenti strumenti grafici, spesso di una GPU (*Graphics Processing Unit*) separata in grado di gestire grafica 2D e 3D a livelli davvero notevoli. La maggior parte delle GPU supporta l'API OpenGL, e i programmatori in Python possono accedere a questa API attraverso pacchetti esterni. Nel Capitolo 8, vedremo come utilizzare OpenGL per la grafica 3D.

Questo libro è stato scritto allo scopo di mostrare come scrivere applicazioni Python migliori, che abbiano buone prestazioni e siano di facile manutenzione, oltre che di facile uso. Presuppone una certa conoscenza della programmazione in Python e va inteso come il tipo di libro a cui rivolgersi dopo aver imparato a programmare in Python, sfruttando la documentazione o altri libri – come *Programming in Python 3, Second Edition* (cfr. la bibliografia a fine libro per i dettagli). Questo libro intende fornire idee, ispirazione e tecniche pratiche che aiutino i lettori a raggiungere un livello più alto nella programmazione in Python.

Tutti gli esempi riportati sono stati testati con Python 3.3 (e ove possibile con Python 3.2 e Python 3.1) su Linux, OS X e Windows (nella maggior parte dei casi). Sono disponibili per il download presso il sito web dell'edizione inglese del libro, all'indirizzo http://www.qtrac.eu/pipbook.html, e dovrebbero funzionare con tutte le future versioni di Python 3.x.