



UNIMORE

UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Corso di
FONDAMENTI DI PROGRAMMAZIONE

Problem Decomposition

Prof. Marco Mamei

Problem Decomposition

- Problem decomposition involves starting with a large problem that may not be fully specified and breaking it down into subproblems, each of which is well-defined and useful for solving our overall problem. Our goal is then to write a function to solve each of those subproblems.
- The process of starting with a large problem and breaking it down is called ***problem decomposition***. The way we're doing this here is synonymous with the software engineering technique known as ***top-down design***

Problema e Requisiti

- **Esempio:** Gestore prestiti di una piccola biblioteca
- **Problema e Requisiti**
 - Utenti prendono in prestito libri; ogni libro a un solo utente
 - Prestiti hanno scadenze e penali
 - Funzionalità: ricerca, registrazione utenti, prestito/restituzione, calcolo penali
 - Non funzionali: semplicità, testabilità, dati in memoria

Decomposizione

- **Dati:** catalogo, utenti, prestiti

```
# crea un dizionario per memorizzare i dati di libri di una biblioteca  
# ogni libro ha un id, un titolo, un autore e l'informazione sul fatto  
che sia in prestito o meno
```

```
# crea un dizionario per memorizzare i dati degli utenti della  
biblioteca
```

```
# crea un dizionario per memorizzare i prestiti con la data di  
prestito e restituzione e gli id della persona e del libro
```

- **Gestione dati:** aggiungi_libro, aggiungi_utente, cerca_libro, cerca_utente
- **Business logic:** presta_libro, restituisci_libro

Funzione restitisci_libro

- restitisci_libro
 - Pre-condizioni
 - esiste_libro(book_id)
 - libro_in_prestito(book_id)
 - Sotto funzioni
 - cerca_presito
 - calcola_penale_restituzione(book_id, data_presito, data_)

Casi limite ed Estensioni

- Errori: libro inesistente, utente sconosciuto, prestito doppio, restituzione senza prestito
- Estensioni: limite prestiti per utente, prenotazioni, persistenza JSON
- Testing con doctest sulle funzioni pure

Linee guida per la Decomposizione dei Problemi

- **Dividi e conquista**
 - Spezza il problema complesso in sottoproblemi semplici e gestibili.
- **Responsabilità singola**
 - Ogni funzione dovrebbe fare *una sola cosa* e farla bene.
- **Nomina significativa**
 - Dai a funzioni e variabili nomi chiari e autoesplicativi.
- **Riutilizzo e testabilità**
 - Scrivi funzioni piccole e indipendenti: più facili da testare e riutilizzare.
- **Astrazione e livelli**
 - Pensa per livelli: dal flusso principale → a funzioni intermedie → a funzioni di dettaglio.

Moduli

- Un modulo in Python è un file che contiene definizioni di funzioni, variabili o costanti, e che può essere riutilizzato in altri programmi tramite l'istruzione `import`
- In pratica, un modulo serve per organizzare il codice in parti più piccole e leggibili, favorendo il riuso e la manutenzione.
- I moduli possono essere organizzati insieme in cartelle chiamare **package**
- Perché dividere un programma in moduli?
 - Programma più chiaro e leggibile
 - Ogni file ha una responsabilità specifica
 - Riutilizzo del codice in più parti
 - Facilita la manutenzione e i test
- Come importare le funzioni
 - **import nome_modulo**
 - invoco la funzione con: `nome_modulo.funzione()`
 - **from nome_modulo import funzione**
 - invoco la funzione con `funzione()`

Problema Lanciare i moduli da soli

- Vedi esempio di codice
- - Per test: usare `if __name__ == '__main__'`
- Meglio eseguire sempre dal livello root
 - `python modulo1.py`:
 - → Esegue un singolo file isolato
 - → Non riconosce i package
- - `python -m package.modulo1`:
- → Esegue il modulo all'interno del package
- → I relative import funzionano

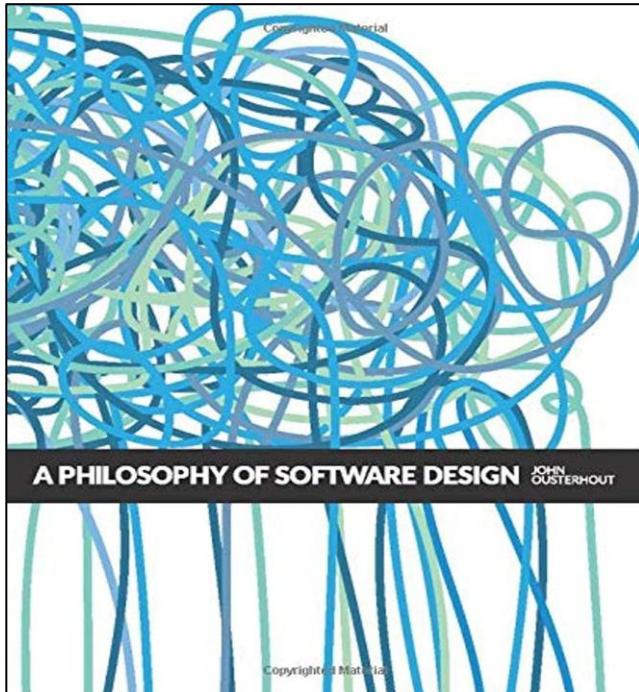
Verso la Programmazione ad Oggetti

- Un caso particolare di un modulo è quello in cui:
 - Una funzione del modulo crea una struttura dati (es. un dizionario)
 - Altre funzioni operano su quel dizionario (sfruttando il passaggio di parametri per riferimento)
 - In questo modo tutte le operazioni per «maneggiare» quella struttura dati sono confinate nel modulo stesso.
- Ci avviciniamo molto alla programmazione ad oggetti

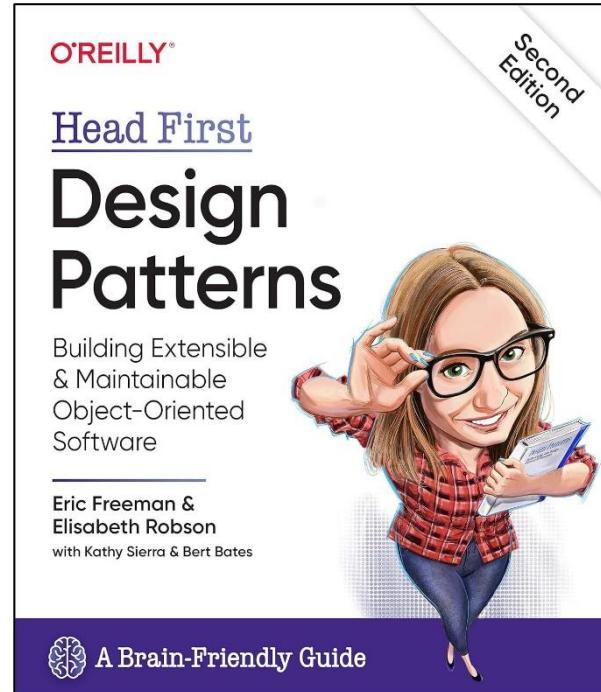
Verso la Programmazione ad Oggetti

- La programmazione orientata agli oggetti (OOP, Object Oriented Programming) è un paradigma di programmazione che permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso l'invocazione di metodi (funzioni).
- La programmazione ad oggetti prevede di raggruppare in una zona circoscritta del codice sorgente (chiamata classe – il modulo nel nostro caso) la dichiarazione delle strutture dati e delle procedure che operano su di esse.
- Le classi, quindi, costituiscono dei modelli astratti, che a tempo di esecuzione vengono invocate per istanziare o creare oggetti software relativi alla classe invocata. Questi ultimi sono dotati di attributi (dati) e metodi (procedure) secondo quanto definito/dichiarato dalle rispettive classi.
- Tra i vantaggi della programmazione orientata agli oggetti:
 - fornisce un supporto naturale alla modellazione software degli oggetti del mondo reale o del modello astratto da riprodurre
 - permette una più facile gestione e manutenzione di progetti di grandi dimensioni
 - l'organizzazione del codice sotto forma di classi favorisce la modularità e il riuso di codice

Per approfondire



A Philosophy of Software Design
by John Ousterhout



Head First Design Patterns:
Building Extensible and
Maintainable Object-Oriented
Software by Eric
Freeman, Elisabeth Robson