**✅ METODOLOGIA TOP-DOWN E BOTTOM-UP**

🔝 Approccio Top-Down

L'approccio top-down è una metodologia di progettazione e sviluppo in cui si parte da una **visione globale del problema** e si procede suddividendolo progressivamente in **sottoproblemi sempre più specifici e dettagliati**, fino ad arrivare a blocchi elementari risolvibili facilmente.

**Caratteristiche:**

* Si basa sull’**astrazione**: prima si pensa “in grande”, poi si entra nei dettagli.
* Consente di scrivere una prima struttura generale del programma (una sorta di scheletro) e poi di riempire man mano le varie parti.
* Promuove la **modularità**: ogni parte del sistema viene progettata come un modulo o sottoprogramma indipendente.

**Vantaggi:**

* Maggiore **leggibilità** e **manutenibilità** del codice.
* Ogni sottoprogramma può essere testato separatamente.
* Facilita la collaborazione in team (ognuno lavora su una parte).

**Esempio:**

Progettare un’applicazione per un e-commerce.

* Fase 1: Identificare le macro-funzioni (gestione prodotti, utenti, ordini, pagamenti).
* Fase 2: Per ogni macro-funzione, definirne i sottocomponenti (es. per ordini: creare ordine, annullare ordine, visualizzare ordine).
* Fase 3: Implementare ogni funzione con codice specifico.

### La metodologia top-down nella progettazione del software è un approccio che si concentra sulla **decomposizione di un sistema complesso in parti più piccole e gestibili**, procedendo da una visione generale verso i dettagli implementativi. Immagina di disegnare un albero: inizi con il tronco (l'idea generale del software), poi passi ai rami principali (le funzionalità chiave), e infine arrivi alle foglie (i singoli moduli e le funzioni).

### Ecco i concetti chiave e i vantaggi di questo approccio:

### **Come funziona:**

### **Identificazione dei requisiti di alto livello:** Si parte definendo gli obiettivi generali e le funzionalità principali del software. Cosa deve fare il sistema nel suo complesso?

### **Decomposizione funzionale:** La funzionalità principale viene suddivisa in sottosistemi o moduli più piccoli e indipendenti. Questo processo continua gerarchicamente, scomponendo ogni modulo in componenti ancora più piccoli, fino a raggiungere un livello di dettaglio tale da poter essere implementato direttamente con codice.

### **Progettazione dell'interfaccia:** In ogni fase di decomposizione, si definiscono le interfacce tra i diversi moduli. Come comunicheranno tra loro le diverse parti del sistema? Quali dati si scambieranno?

### **Implementazione e test:** Una volta raggiunti i moduli di basso livello, si procede con la codifica e il testing individuale di ciascun componente.

### **Integrazione:** Infine, i moduli testati vengono integrati gradualmente, partendo dai livelli più bassi fino a ricostruire l'intero sistema.

### **Vantaggi della metodologia top-down:**

### **Chiarezza e comprensione:** La visione generale del sistema rimane sempre al centro, facilitando la comprensione della sua architettura e delle relazioni tra le diverse parti.

### **Gestione della complessità:** Dividere un problema grande in problemi più piccoli rende la progettazione e lo sviluppo più gestibili e meno soggetti a errori.

### **Identificazione precoce dei requisiti:** Concentrandosi inizialmente sulle funzionalità di alto livello, è più facile individuare eventuali lacune o incongruenze nei requisiti.

### **Facilità di manutenzione:** Un'architettura ben definita e modulare rende il software più facile da comprendere, modificare e aggiornare nel tempo.

### **Migliore gestione del progetto:** La suddivisione in compiti più piccoli facilita la pianificazione, l'assegnazione delle risorse e il monitoraggio dei progressi.

### **Svantaggi e considerazioni:**

### **Difficoltà nella definizione iniziale:** Può essere difficile definire con precisione i requisiti di alto livello all'inizio del progetto, soprattutto se il dominio applicativo è nuovo o poco conosciuto.

### **Rischio di decisioni premature:** Le decisioni prese nelle fasi iniziali, basate su una comprensione ancora parziale dei dettagli, potrebbero rivelarsi inadeguate in seguito.

### **Integrazione complessa:** Se le interfacce tra i moduli non sono ben definite, l'integrazione finale può diventare problematica.

### **Poca flessibilità ai cambiamenti:** Apportare modifiche significative all'architettura nelle fasi avanzate del progetto può essere costoso e complesso.

### **In sintesi:**

### La metodologia top-down è un approccio classico e strutturato per la progettazione del software, particolarmente efficace per progetti di medie e grandi dimensioni dove è fondamentale avere una chiara visione d'insieme fin dall'inizio. Tuttavia, è importante bilanciare la pianificazione iniziale con una certa flessibilità per adattarsi a eventuali cambiamenti che possono emergere durante il ciclo di vita del progetto.

🔽 Approccio Bottom-Up

Il bottom-up è l’opposto del top-down. Si inizia **dalla realizzazione delle componenti più semplici e generiche** (come funzioni riutilizzabili o librerie), per poi combinarle in strutture via via più complesse fino ad arrivare al sistema completo.

**Caratteristiche:**

* Si costruisce il sistema **dalle fondamenta**.
* Utilizza spesso **funzioni preesistenti** o ben collaudate.
* Spesso impiegato quando si lavora con **API, librerie o moduli già disponibili**.

**Vantaggi:**

* Riduce la duplicazione del codice.
* Riutilizzo di componenti affidabili e già testati.
* Permette di sviluppare **blocchi funzionali riutilizzabili** in più progetti.

**Esempio:** Creo prima un modulo per calcolare le tasse → poi un modulo per il carrello → poi li integro in un sistema di e-commerce.

**🧩 SOTTOPROGRAMMI**

Un **sottoprogramma** (anche detto procedura o funzione) è una porzione di codice progettata per svolgere un compito preciso, che può essere richiamata da più punti del programma.

Tipologie:

* **Funzione**: ritorna un valore (es. int somma(int a, int b)).
* **Procedura**: non ritorna nulla (in C void, in Python è una funzione che non usa return, ecc.).

Vantaggi:

* Riduce la **ridondanza** (scrivi una volta, usi più volte).
* Migliora l’organizzazione del codice.
* Facilita la **manutenzione e il debugging**.

**🔣 PARAMETRI**

Parametri formali:

* Sono quelli dichiarati nella **definizione** del sottoprogramma.
* Esistono solo all’interno del blocco del sottoprogramma.

Parametri attuali:

* Sono i valori **effettivi** passati al momento della chiamata.

**Modalità di passaggio:**

* **Per valore**: si passa una copia → il valore originale non viene modificato.
* **Per riferimento**: si passa direttamente la variabile → il valore originale **può essere modificato**.

**🧠 VARIABILI LOCALI**

* Le variabili dichiarate all’interno di un sottoprogramma sono **locali**.
* **Visibilità limitata**: non possono essere usate fuori dal blocco.
* Vengono create quando il sottoprogramma viene eseguito e **distrutte quando termina**.

Vantaggi:

* Evitano conflitti con variabili globali.
* Mantengono i dati **incapsulati**, migliorando l’affidabilità.

**🔁 RICORSIONE**

La **ricorsione** è una tecnica in cui un sottoprogramma **chiama se stesso** per risolvere un problema.

Struttura di base:

1. **Caso base**: condizione che interrompe la ricorsione.
2. **Chiamata ricorsiva**: il sottoprogramma si richiama con un parametro più semplice.

**Esempio:**

def fattoriale(n):

if n == 0:

return 1

else:

return n \* fattoriale(n - 1)

Quando usarla:

* Quando il problema può essere definito in **termini di sé stesso** (es. alberi, strutture nidificate, problemi matematici).
* Quando la versione iterativa sarebbe più complicata da scrivere o capire.

Svantaggi:

* **Consuma più memoria** (usa lo stack delle chiamate).
* Può portare a **stack overflow** se il caso base non è ben definito.