Approcci nella soluzione di problemi

Top-down

Una metodologia di trattamento dei problemi per *scomposizione* → *sviluppo degli algoritmi per* ***raffinamenti successivi.***

Dal problema complesso …

… scomponendo in sottoproblemi più semplici …

si risolve un problema elementare spesso di soluzione già nota

Nella **scomposizione** dei problemi in sotto-problemi, **risolvendo per approfondimenti successivi**, possiamo riferirci mediante un ***nome*** ad blocco di comandi (*direttive*) e proporre una soluzione "**modulare**" più chiara.

Ad esempio si usa lo pseudocodice per indicare in **sequenza** i tre *moduli* essenziali nella soluzione e che analizzeremo dopo, individuandone i singoli passi: un modulo di *inizializzazione* dei dati utili (col nome **inizio**), un'elaborazione col nome **elabora** ed un modulo per *visualizzare a monitor* (CLI) il risultato dell'elaborazione col nome **visualizza**:

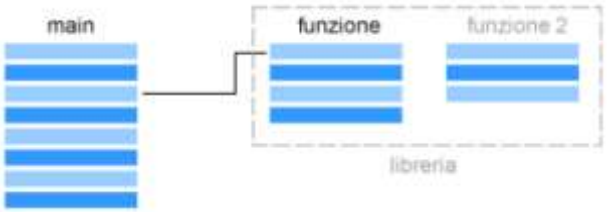
{ **inizio** **elabora** **visualizza** }	*Potendo passare dati da elaborare* { ***k*** ← valore **elabora (*k*)** **visualizza** }

Progettare soluzioni **modulari**, serve per *ridurre la complessità* di un problema consentendo facile implementazione con approccio *top*-down, lavorando in *equipe*.

Progettare bene un **modulo** significa pensarlo con le seguenti caratteristiche:

- efficienza,
- generalizzazione
- portabilità

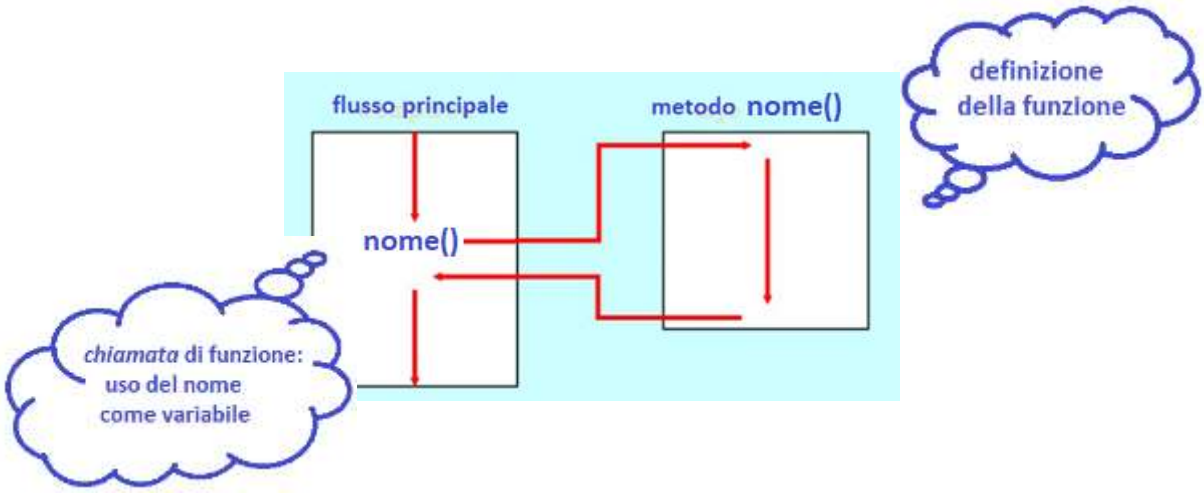
Implementeremo tale modulo come **funzione**. Una **funzione** è un blocco di comandi associato ad un **nome che produce sempre un risultato**. Le funzioni possono essere *definite* e *invocate.*



Scomposizione in moduli: concetto di funzione

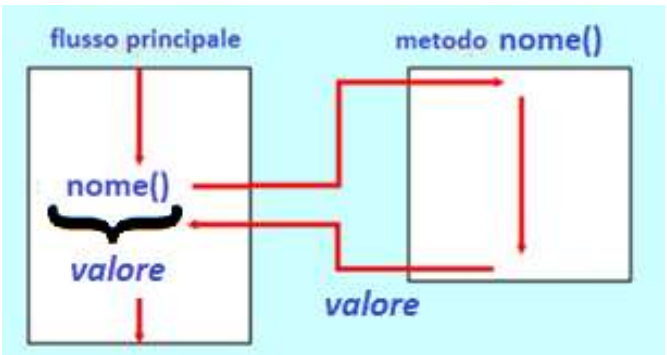
Una **funzione** è un *modulo di programma* (sotto-insieme di istruzioni o segmento di programma detto *corpo*), associato ad un **nome**, che può essere inteso come una "scatola nera" capace di realizzare una specifica elaborazione fornendo sempre un **risultato** (se la funzione non restituisce un valore è detta *void*) potendo utilizzare dati che le vengono **passati**, ad esempio come valori, al momento della *chiamata* da programma

Tale "scatola" non può essere inserita in un'altra "scatola" ma può essere usata trattando il suo ***nome*** come una ***variabile***.



Il **nome** della funzione è l'**indirizzo** dove, in RAM, è allocata la *prima istruzione* del segmento di programma.

Come per ogni variabile, anche per una funzione è necessario definirne il **tipo**: si deve informare il compilatore affinché riservi l'area di memoria necessaria a contenere il risultato della funzione che poi si *sostituisce al nome*, al ritorno dalla chiamata.



Se la funzione *chiamata* non ritorna un valore alla funzione *chiamante*, in realtà il risultato risulta indefinito ed è di tipo *void*

Motivazione all'uso

Una funzione è un modo di *arricchire il linguaggio*.

Le funzioni possono essere raggruppate in **librerie** e ***riutilizzate*** da altri programmi;

rendono un programma più immune da errori imprevisti grazie alla possibilità di *nascondere* i dati

prevedendo meccanismi di **passaggio controllato**

delle informazioni;

consentono una scrittura più semplice e aumentano la *leggibilità* del programma.

Una **funzione**, infatti, serve per *ridurre la complessità* di un problema consentendo facile implementazione di soluzioni con approccio *top-down* che prevedono un lavoro parallelo in *equipe*.

Progettare bene una **funzione** significa pensarla con le seguenti caratteristiche: *efficienza*, *generalizzazione* e *portabilità*.

L'**efficienza** si valuta in base all'*utilizzo* che la funzione fa *delle risorse* del calcolatore: CPU (tempo), memoria RAM (spazio).

Con **portabilità** si intende *usabilità in ambienti di esecuzione diversi* e più generale l'indipendenza - in fase di esecuzione - dalla piattaforma hardware e/o software di esecuzione.

Con **generalizzazione** si intende *applicabile*, spesso arbitrariamente, alla maggior parte o alla totalità dei casi.