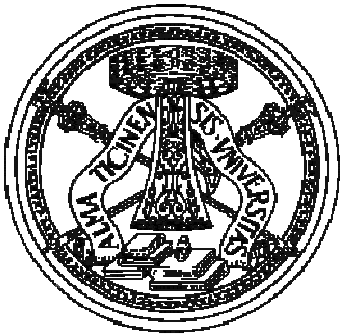
UNIVERSITÀ DEGLI STUDI DI PAVIA
FACOLTÀ DI INGEGNERIA



Introduzione alla programmazione

Riferimenti

- Emanuele Goldoni
  - Laboratorio Reti (MN)
    - Tel. 0376-286234
  - E-mail: emanuele.goldoni@unipv.it
- Slide sul sito del corso
  - http://gamma.unipv.it/elementi_informatica/
- Ricevimento
  - Dalle 10:00 alle 11:00 (nei giorni dei seminari)
  - Laboratorio di Telecomunicazioni (Pavia – Facoltà di Ingegneria - Piano H)

Obiettivo dei seminari

- Fornire gli strumenti di base per riuscire ad affrontare i problemi in modo rigoroso e a tradurre gli stessi in programmi
- Introdurre un semplice linguaggio di programmazione
- Illustrare alcuni tra i più comuni algoritmi

Contenuti

- Introduzione alla programmazione
- Introduzione a Matlab
- Elementi del linguaggio di scripting di Matlab
- Utilizzo di Matlab in Ingegneria
- Algoritmi di ricerca
- Algoritmi di ordinamento
- Complessità degli algoritmi

Risolvere un problema

- Per risolvere un problema si procede innanzitutto all'individuazione:
  - Delle informazioni, dei *dati* (noti)
  - Dei *risultati* (incogniti) desiderati
- Il secondo passo consiste nell'individuazione di un *procedimento* adeguato
- Si *scompone* infine il procedimento con l'individuazione delle operazioni necessarie per il calcolo di *risultati intermedi*

Dal problema al programma

- Per realizzare un programma che risolva un problema è necessario quindi:
  - Individuare i dati in ingresso e in uscita
  - Individuare un procedimento adeguato
  - Scomporre il procedimento in una sequenza di azioni elementari ed univoche

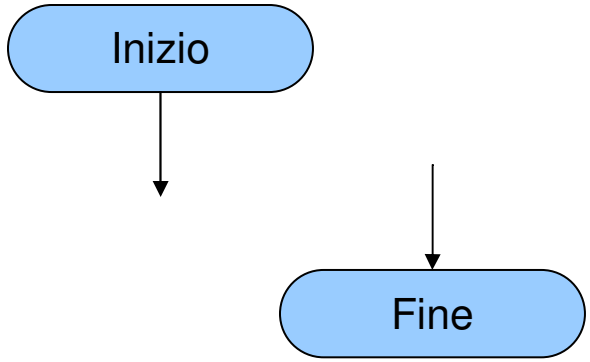
- Per meglio affrontare l'ultimo punto, si fa spesso ricorso ai diagrammi di flusso (o flow-chart)

Flusso di esecuzione (1)

- Il flusso di esecuzione può essere rappresentato graficamente con un *diagramma di flusso*
  - Si sviluppa graficamente su due dimensioni
  - Elimina le ambiguità utilizzando una logica di selezione ambivalente e linee di flusso univoche
  - Si basa su pochi simboli
  - E' un linguaggio universale

Flusso di esecuzione (2)

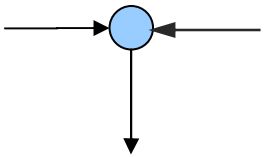
- Innanzitutto un programma, e di conseguenza una sua rappresentazione grafica, deve sempre avere un inizio ed una fine



- Tra l'inizio e la fine vi deve sempre essere almeno una istruzione

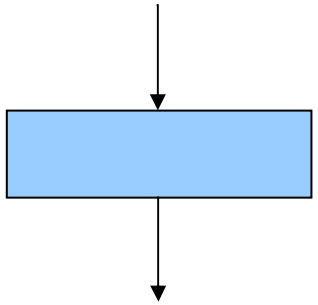
Flusso di esecuzione (3)

- La risoluzione di un problema consiste nell'esecuzione ordinata di una *sequenza* di operazioni
  - L'ordine nell'esecuzione delle istruzioni è fondamentale
  - Nei *flow-chart* è garantito dall'orientamento delle frecce che collegano i blocchi



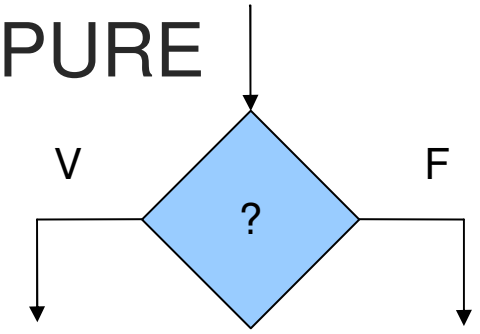
Flusso di esecuzione (4)

- A volte le azioni vanno eseguite “sempre”. In tal caso diciamo che si tratta di operazioni *non condizionali*



Flusso di esecuzione (5)

- Possono essere presenti istruzioni *condizionali*, la cui esecuzione dipende cioè da *scelte* effettuate in base ai dati
  - Concettualmente, possiamo immaginare che il flusso di esecuzione si *ramifichi* (in base ad una condizione decido se eseguire un'operazione OPPURE un'altra)



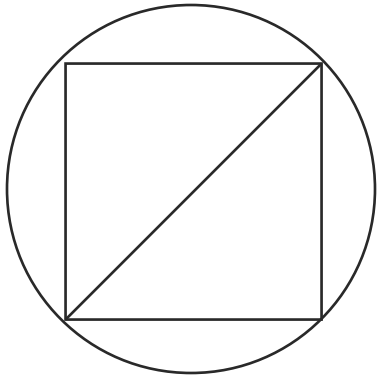
Diramazione (condizionale)

Algoritmi

- Per indicare la scomposizione del problema viene spesso utilizzato il termine *algoritmo*
  - "*Insieme finito di istruzioni elementari univocamente interpretabili che, eseguite in un ordine stabilito, forniscono la soluzione di un problema in un numero finito di passi*"
- I *flow-chart* sono un linguaggio utile per descrivere gli algoritmi

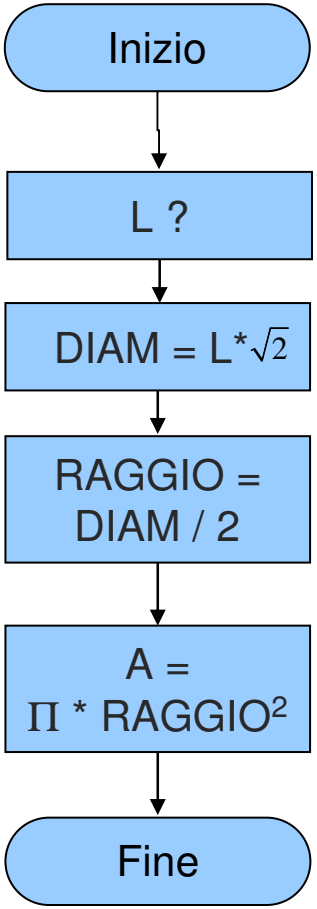
Area del cerchio circoscritto (1)

- *Determinare l'area A del cerchio circoscritto ad un quadrato avente il lato di misura L*



- Soluzione
  1. diam = diag = **L*** sqrt(2)
  2. raggio = diam / 2
  3. **A** = Π * raggio * raggio

Area del cerchio circoscritto (2)



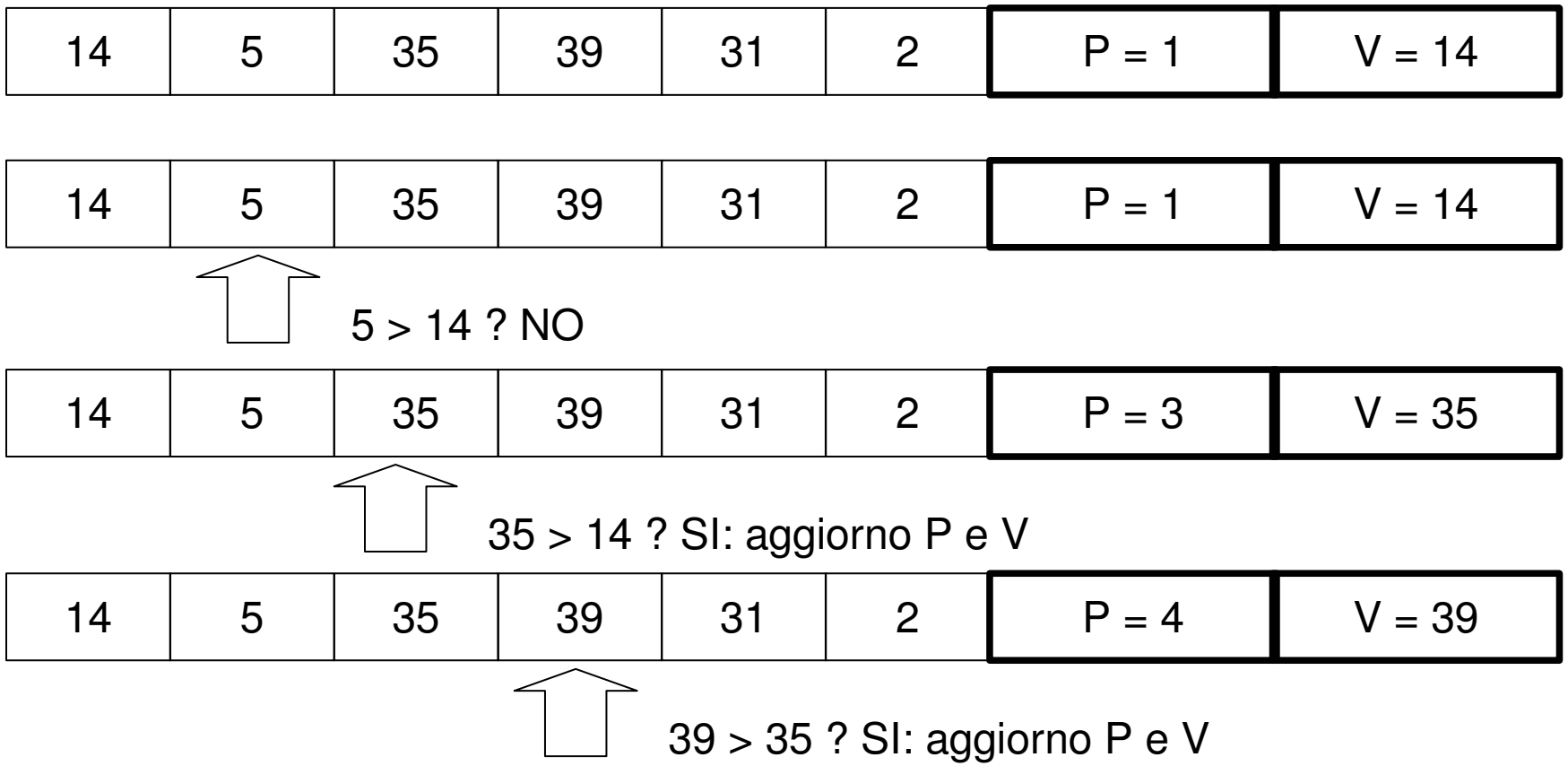
Ordinamento lista di numeri (1)

- *Ordinare una lista di numeri dati*
- Le operazioni disponibili
  - Scambiare la posizione di due numeri
  - Confrontare due numeri
  - Annotare valore e posizione di un numero
- Soluzione:
  - Scorro la lista, cerco il maggiore e lo sposto in fondo; ripeto quindi l'operazione per trovare il penultimo e così via…

Ordinamento lista di numeri (2)

1. Scorro la lista dalla prima posizione fino a n = ultima posizione
2. Segno valore e posizione del primo numero (posizione=1, valore=__)
3. Confronto il valore segnato con quello nella posizione 2
4. Se è maggiore di quello segnato, aggiorno valore e posizione
5. Ripeto i punti 3 e 4 per le posizioni 2, 3 .. n etc fino a n
6. Diminuisco n di una unità
7. Se n è maggiore di due, torno al punto 2; altrimenti esco

Ordinamento lista di numeri (3)



Ordinamento lista di numeri (4)

Dopo tutti i confronti...

14	5	35	39	31	2	P = 4	V = 39

1° scambio e ricomincio

14	5	35	**2**	31	**39**	P = 1	V = 14

Dopo il secondo giro di confronti...

14	5	35	2	31	39	P = 3	V = 35

2° scambio e ricomincio

14	5	**31**	2	**35**	39	P = 1	V = 14

E così via...

Scelta dell'algoritmo

- Alcune considerazioni utili
  - Un algoritmo semplice può risultare meno efficace, specialmente in termini di velocità
    - L'efficienza può consentire un risparmio di risorse
  - Non necessariamente un algoritmo "furbo" è più efficiente o affidabile
    - La semplicità può facilitare la comprensione e la correzione

Linguaggi (1)

- Un linguaggio di programmazione è un linguaggio formale costituito da simboli (*alfabeto*), combinati in sequenze in base a regole precise (*sintassi*) e a cui viene associato un determinato significato (*semantica*)
- Le famiglie di linguaggi
  - Linguaggi assembler
  - Linguaggi funzionali
  - Linguaggi imperativi (BASIC)
  - Linguaggi a oggetti (Java) etc…

Linguaggi (2)

- La macchina utilizza esclusivamente l'alfabeto binario
  - I *linguaggi* consentono quindi di descrivere le operazioni da far compiere al calcolatore in maniera più semplice
- E' necessaria un'operazione di traduzione dal linguaggio di programmazione al codice comprensibile dal calcolatore, effettuata per mezzo di opportuni programmi
- Questi traduttori si dividono in due categorie: *compilatori* e *interpreti*

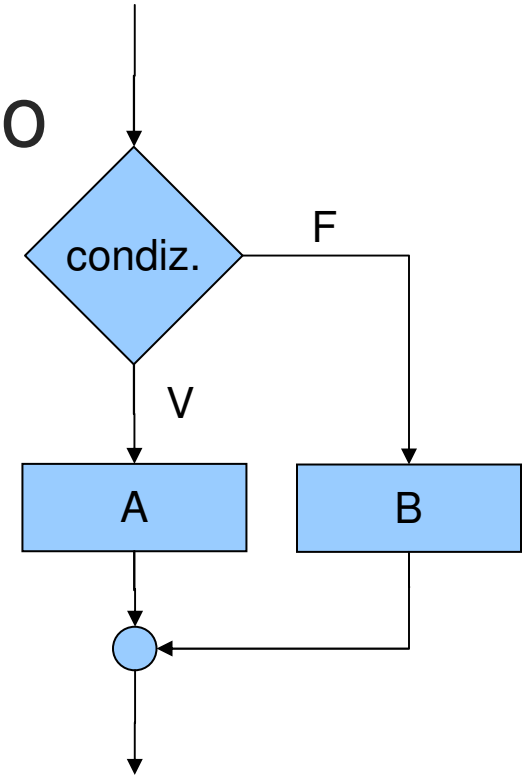
Costrutti di programmazione

- Esistono alcuni costrutti di base tipici dei linguaggi di alto livello
- Qualsiasi algoritmo rappresentabile con un flow-chart strutturato può essere tradotto in un programma utilizzando appena tre strutture:
  - *Sequenza*
  - *Selezione*
  - *Iterazione*

Selezione

- È un costrutto tipico, del tipo

```
if (condizione) then
      fare A
[else
      fare B
end]
```



- Si usa per scegliere tra due alternative
  - Il blocco `else` è opzionale

Iterazione (1)

- Per poter ripetere lo stesso gruppo di istruzioni più volte, è necessario utilizzare un altro costrutto: il ciclo (o *loop*)
- E' necessario definire una *condizione* per stabilire quante volte ripetere il blocco di istruzioni
  - Esistono due costrutti che utilizzano criteri duali per interrompere l'iterazione

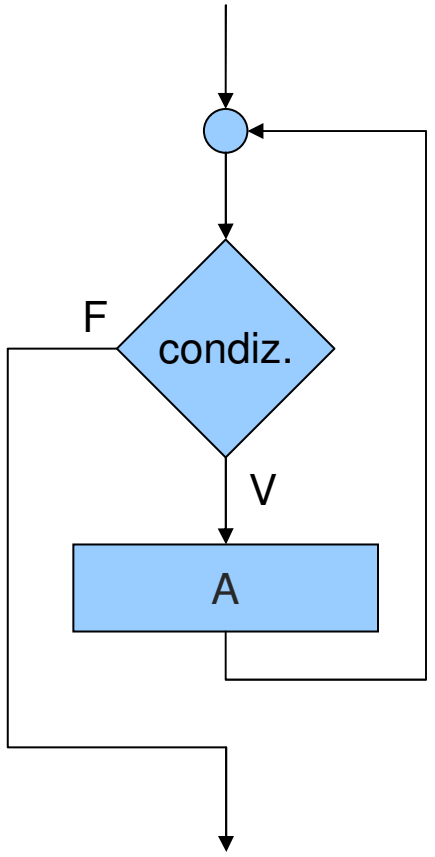
Iterazione (2)

- Un blocco può essere eseguito “fintanto che” una certa condizione è verificata ed interrompere l’esecuzione non appena questa diventa falsa
- Se allo stato iniziale la condizione è falsa, le istruzioni non vengono mai eseguite
  - Questo è detto ciclo ad *ingresso controllato*

Iterazione (2)

- Il costrutto *while* realizza un ciclo ad ingresso controllato

```
while (condizione)
      fare A
```



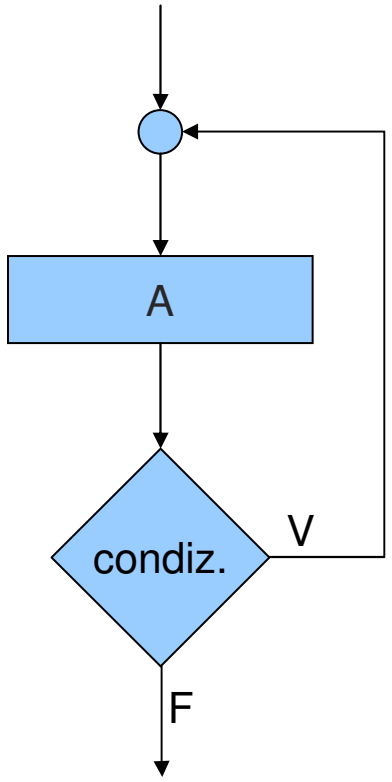
Iterazione (3)

- Un blocco può essere eseguito anche “fino a quando” non si verifica una condizione prestabilita
  - Questo è detto ciclo ad *uscita controllata*
- In questo caso prima viene eseguita un’iterazione e poi controllato il valore di verità della condizione

Iterazione (4)

- Il costrutto *do-while* (o *repeat-until*) realizza un ciclo ad uscita controllata

```
do
   fare A
while (condizione)
```



Iterazione (5)

- Un terzo costrutto viene impiegato se è noto a priori il numero di iterazioni da eseguire:

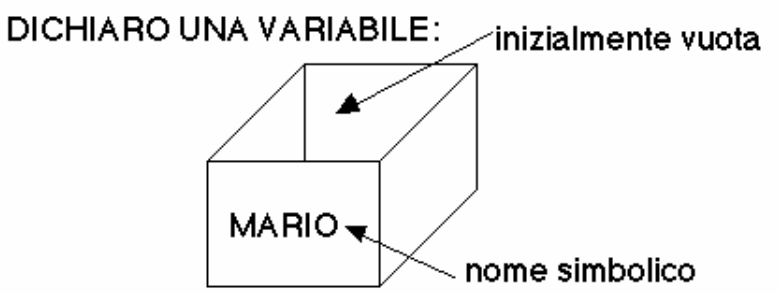
```
for (Val_iniziale; Cond; Incremento)
    fare A;
```

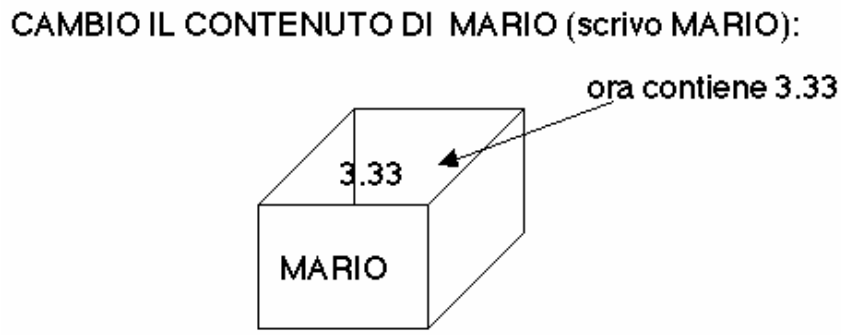
- Semplifica la scrittura e la semplicità dei programmi
- E' una struttura ridondante
  - È comunque riconducibile alle altre due viste in precedenza

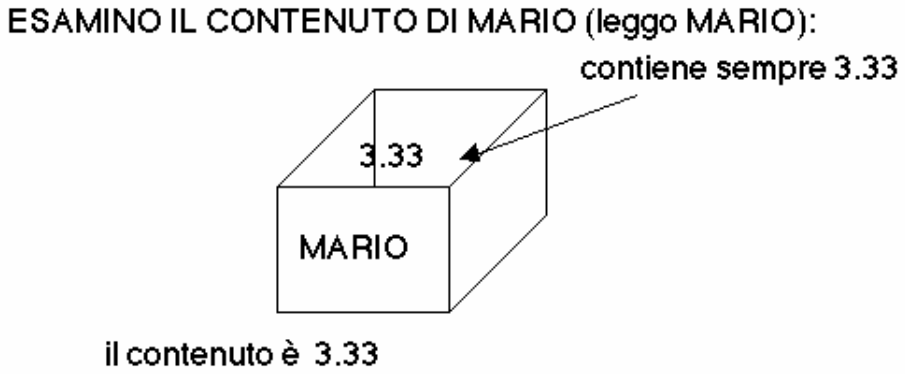
Informazioni nel calcolatore

- I calcolatori utilizzano esclusivamente numeri, mentre il linguaggio (ed il cervello umano) gestisce più facilmente i nomi
- I linguaggi di programmazione permettono quindi di fare riferimento ai dati utilizzando dei nomi, detti *variabili*.
- Una variabile può essere vista come un contenitore avente un nome ed un contenuto
  - Il nome variabile vuole enfatizzare il fatto che il contenuto può cambiare.

Metafora delle variabili







Tipi dato (1)

- Il contenuto di una variabile all'interno del calcolatore è una sequenza di bit
  - occorre determinare quale valore contiene la variabile, cioè come interpretare tali bit.
- I linguaggi di programmazione associano ad ogni variabile informazioni anche sul tipo di valore che essa contiene.
- L'attribuzione di un *significato* particolare alla sequenza binaria è fatto attraverso il concetto di *tipo di dato*

Tipi dato (2)

- Un insieme minimo di *tipi di dato* che ci interessa risulta composto da:
  - numeri *interi*
  - numeri in *virgola mobile*
  - *caratteri*
    - La stringa può essere vista come una sequenza di caratteri
- Normalmente sono disponibili strumenti che consentono di *convertire* un dato da un tipo ad un altro

Vettori

- Una variabile contiene solo un dato mentre molti algoritmi gestiscono una serie di dati ordinati e dello stesso tipo.
- In questo caso si utilizzano i *vettori,* un insieme ordinato di variabili dello stesso tipo
- Per accedere ad un elemento specifico se ne indica la posizione
- Esempio:
  - a = [1, 4, 33, 12, 42, 66]
    - a(2) = 4, a(5) = 42 etc…

Matrici

- È possibile estendere i vettori al caso bidimensionale per ottenere delle matrici (“vettore di vettori”)
  - Per identificare un elemento se ne specifica la posizione (riga, colonna)
- Si può anche lavorare con vettori n-dimensionali
  - Anche se possiamo aver difficoltà a “visualizzarli”, da un punto di vista formale non presentano difficoltà

Funzioni (1)

- Capita spesso che una parte dell'algoritmo che si vuole implementare sia
  - indipendente dall'algoritmo specifico (cioè ha senso anche preso da solo)
  - utilizzabile in più situazioni
- In questi casi è utile raggruppare il codice relativo in un'unica entità, nota come *funzione*
- Una volta scritta la funzione, il codice che la compone può essere semplicemente *chiamato* (o *invocato*) quando necessario

Funzioni (2)

- Una funzione è caratterizzata da
  - un nome
  - un insieme di argomenti (dati sui quali la funzione opera)
  - un tipo di dato restituito (la funzione è *valutata* e si ottiene un certo tipo di risultato)
- Conoscendo queste tre informazioni è possibile utilizzare una funzione senza necessità di conoscere l'algoritmo o i calcoli effettuati all'interno della funzione stessa
  - *Distinzione tra implementazione e interfaccia*

Riferimenti

- N. Wirth, “Algoritmi + Strutture dati = Programmi”, 1987
- G. Cupini et al., “Corso di Informatica”, 1987
- D. Knuth, The Art of Computer Programming, 1970s