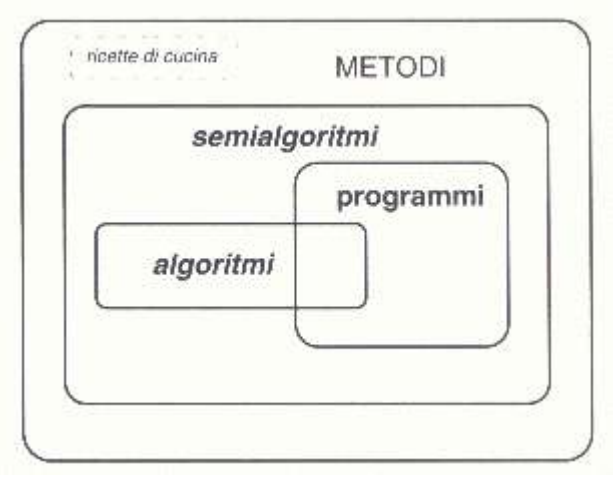
Problemi e programmi

Nel senso più generale del termine, programmare è sinonimo di "***risolvere problemi***" ed è un'attività fortemente creativa.

La soluzione, con riferimento alla sequenza di azioni che si prefiggono il raggiungimento di certi obiettivi precisi, viene espressa in un *linguaggio*: in politica si illustra il programma di un partito (una serie di impegni da mantenere nell'immediato futuro); in un corso di studi si rende nota in forma orale e scritta la sequenza temporale di argomenti da svolgere e di obiettivi didattici da raggiungere; nel mondo dello spettacolo si rende disponibile in forma scritta l'elenco delle varie parti che compongono una rappresentazione o di tutte le manifestazioni che avranno luogo in un certo periodo.

Anche una ricetta di cucina ed uno spartito musicale sono **metodi** risolutivi dove una variazione, anche piccola, della sequenza di azioni da svolgere altera il risultato finale.



Per giungere dal problema ben preciso alla concreta soluzione (programma) occorre innanzi tutto individuare un ***esperto*** che sia in grado di **analizzare** il tipo di problema, individuare una **strategia risolutiva** e formulare una soluzione sotto forma di una **sequenza di azioni** (prodotte da un esecutore capace di eseguire quelle determinate *istruzioni* cioè comandi non ambigui ed espressi nel linguaggio dell'esecutore → ***azione*** = *istruzione* + esecutore).

Dunque, affinché un ***esecutore*** possa interpretare ed eseguire correttamente quella determinata sequenza di indicazioni occorre convertirle in una forma a lui comprensibile: se l'esecutore è un essere umano, il cuoco nell'esempio della ricetta, è sufficiente che l'esperto (il gastronomo) usi il *linguaggio naturale*, prestando attenzione alla sintassi e alla chiarezza espositiva ed, anche se la formulazione è imprecisa e poco dettagliata, il cuoco potrebbe essere in grado di interpretare le istruzioni in modo intelligente, apportando piccole modifiche nell'esecuzione della procedura.

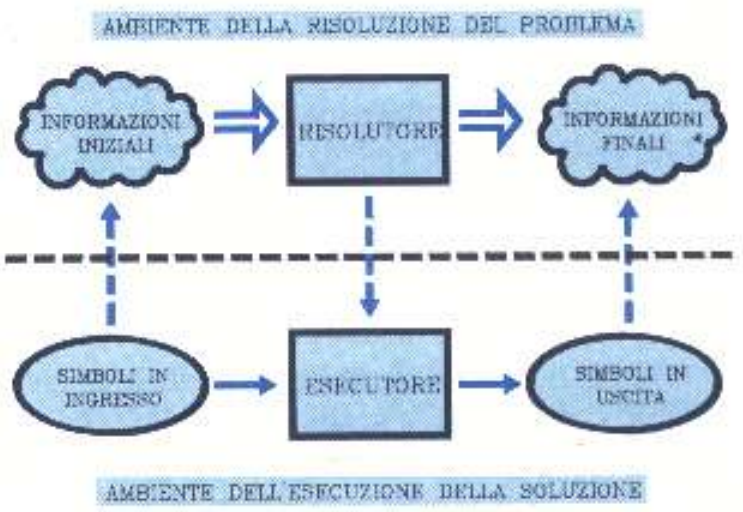
Dagli anni '50, il termine algoritmo[1] è normalmente usato come sinonimo di ***procedura***, ***procedimento***, ***metodo di calcolo*** (non necessariamente numerico) *come soluzione di un determinato problema.*

Con terminologia più appropriata, si può definire **procedura effettiva** (semialgoritmo – metodo computazionale che non richiede la terminazione per tutti i dati in ingresso) cioè **realmente meccanizzabile**, quella che equivale ad un programma in *linguaggio universale* (possibilità di implementazione con qualunque specifico linguaggio di programmazione) nel risolvere problemi dove l'esecutore è un automatismo (automa universale realizzato su modello von Neumann).

> **Tesi di *Church-Turing***
> Una *procedura* è **effettiva** (è un semialgoritmo) se e solo se esiste un programma equivalente ad essa espresso in un *linguaggio universale*.

[1] Termine che trae le sue origini dai metodi usati dai commercianti nel medio evo per eseguire le quattro operazioni sulle rappresentazioni decimali dei numeri (rappresentazione *posizionale*). Metodo alternativo all'uso dell'abaco.

Con il termine ***Algoritmo*** si intende un elenco **finito** di *direttive* interpretabili **univocamente** (senza ambiguità e in modo deterministico cioè non casuale) da un ipotetico esecutore che opera in modo **discreto** (automa che assume stati separati da intervalli di tempo non nulli) e che **termina** in un numero finito di *passi* (*passo* = azione svolta tra uno stato ed il successivo), per tutti i possibili *dati di ingresso* (**complessità limitata** delle istruzioni → effettivamente eseguibili) fornendo i corrispondenti risultati (valore o effetto esterno) richiesti da un dato problema.



L'attenzione alla **formalizzazione** è essenziale quando *risolutore* ed *esecutore sono diversi* e questo comporta che siano di fondamentale importanza, nel ***risolvere un problema***:

- Una precisa **riformulazione**[2] del problema che evidenzi *dati ("datum = fatto")* e *risultati* per evitare ambiguità con attenzione a:
    - chiarire lo **scopo** (soluzione)
    - evidenziare i **dati**: **disponibili** o da **assumere**

- Lo sviluppo di un *modello* delle relazioni tra dati e risultati, costruito sulle conoscenze che già si posseggono cercando di *parametrizzare* il più possibile il problema in modo che la soluzione possa essere utilizzata per risolvere una *molteplicità di problemi particolari*.

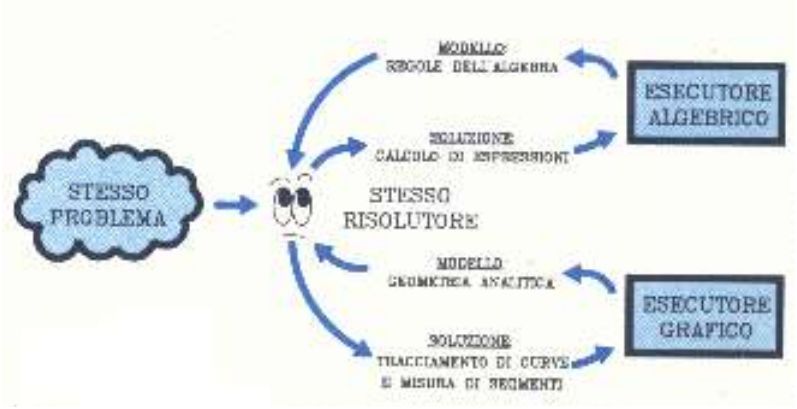
  Attenti sempre al fatto che nel rendere generale si compie un'approssimazione → possibile errore → sarà necessaria una verifica nell'applicazione dello stesso modello ad altri problemi (*validazione*)

Lo sviluppo del processo dai dati ai risultati è doppiamente controllato: i dati di partenza e i modelli adottati forniscono indicazioni su come scegliere i risultati.

Questi ultimi vanno confrontati con i primi per una verifica della coerenza interna della soluzione ottenuta con trattamento automatico.

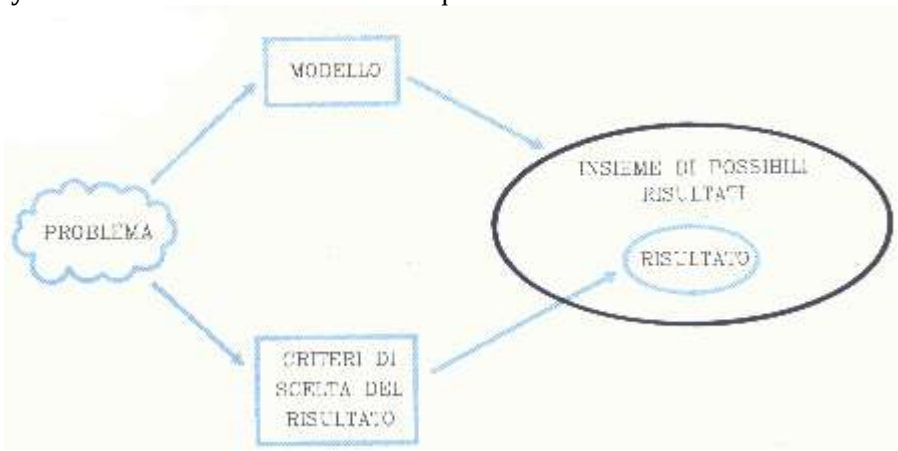
[2] A volte la riformulazione con un percorso logico inverso rispetto a quello dato può facilitare la soluzione.

Nel caso di risoluzione di problemi con metodi che usano strumenti matematici si definirà ***modello matematico*** l'insieme di equazioni che esprimono le relazioni tra dati e risultati.



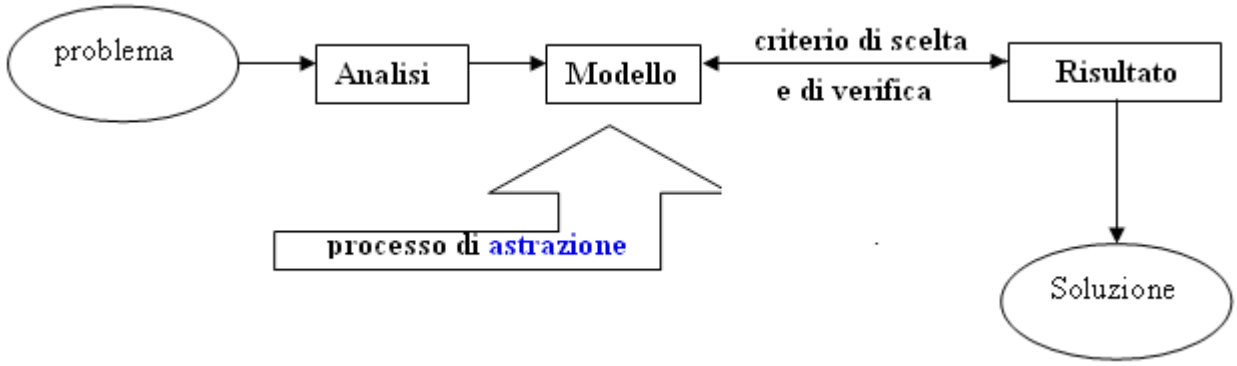
Avendo, ad esempio, il problema di esemplificare il concetto di **diretta proporzionalità**, si potrebbe:

- procedere per via algebrica: i dati vengono codificati con lettere (le incognite) e numeri in notazione scientifica e l' operazione specifica da compiere utilizza ad esempio il seguente modello matematico: $\mathbf{y} = m^{*}\mathbf{x}$ dove m è un numero reale qualsiasi.

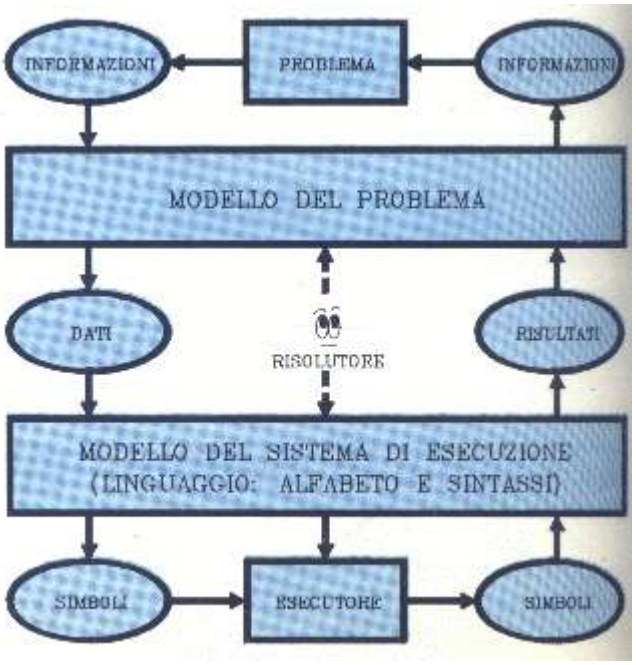


- procedere per via grafica (come modello la geometri analitica) : tracciando nel piano cartesiano una retta ad esempio passante per l'origine.

A seconda del criterio di scelta del risultato si potrà prevedere più opportuna l'una o l'altra modalità ma si dovrà verificare se tale previsione si dimostra corretta ed il risultato ottenuto è quello voluto.



- L'adozione di un *modello* di riferimento che rappresenti l'*esecutore* (specifico linguaggio: alfabeto e sintassi) o il sistema di regole entro cui dovrà svilupparsi la soluzione del problema.

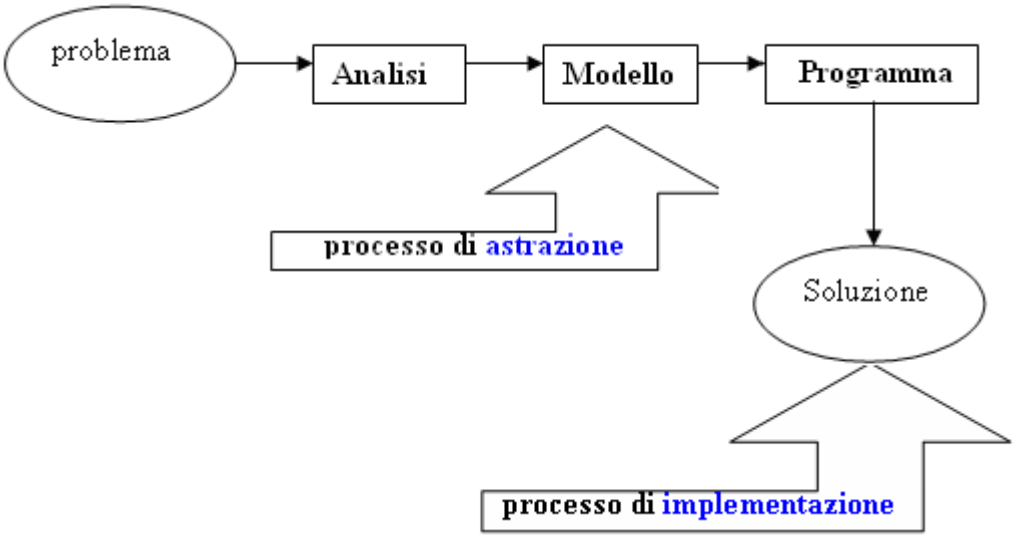


Computer per risolvere problemi

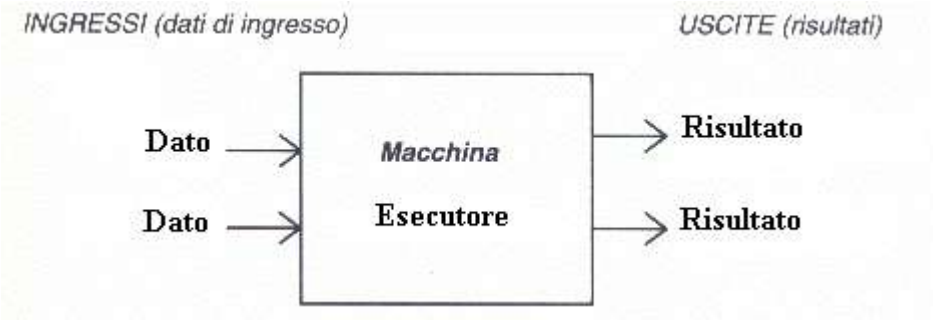
L'uomo ha sempre desiderato uno strumento che potesse rendere automatica la soluzione di problemi, cioè capace di eseguire un suo programma in modo preciso, sostituendosi all'intervento umano.

Il termine **programma**, riferito al mondo dei calcolatori, si usa per intendere una **sequenza di istruzioni** o indicazioni, **scritte in un linguaggio comprensibile al calcolatore** che le interpreta e le esegue per ottenere i risultati richiesti.

In un contesto informatico, la soluzione di un problema può dunque richiedere le **fasi** seguenti:



Nel caso di ***problemi*** risolubili (dal calcolatore) ne possiamo rappresentare la soluzione col diagramma seguente con ricorso al paradigma I/O:



In figura il ***modello del sistema*** (automa esecutore) giustifica anche l'uso dei termini *dati di ingresso* e *dati di uscita* rispettivamente usati per *dati* e *risultati*.

Se l'automa esecutore è il **computer**, potrà essere necessario determinare il **tipo** dei dati con riferimento all'occupazione delle celle di RAM in cui saranno memorizzati i valori ed il ***modello di dati*** più opportuno (semplici o aggregati) in relazione all'efficienza nel recuperare tali valori ed al metodo più o meno sofisticato per l'organizzazione dei dati utilizzato nelle elaborazioni.

E' un ***modello logico***, la descrizione dell'algoritmo risolutivo utilizzando linguaggi non naturali (per ridurne l'ambiguità) ma senza adottare un linguaggio di programmazione specifico (che potrebbe diventare rapidamente obsoleto), allo scopo di progettare/presentare soluzioni per diversi livelli di approfondimento in cui solo l'ultimo livello, quello definitivo, sarà non ambiguo (con metodologia top-down).

Esempio:

Problema degli interessi generalizzato:
Se all'inizio dell'anno avete un certo Debito con la banca e il tasso d'interesse percentuale annuo praticato dalla banca è TassoAnnuo, quanto spendete di interessi ogni giorno ? E ogni mese ?

modello del sistema

modello di dati

variabili di input:

numeri reali: debito, tassoAnnuo

variabili di output:

numeri reali interesseGiornaliero, interesseMensile

dati di lavoro:

numeri reali interesseAnnuo

modello matematico

interesseAnnuo = debito * tassoAnnuo/100 con tassoAnnuo espresso in %
interesseGiornaliero = interesseAnnuo/**365**
interesseMensile = interesseAnnuo/**12**

Algoritmo in pseudocodice - modello logico

{

inizializza debito, tassoAnnuo

interesseAnnuo ← debito * tassoAnnuo/100

interesseGiornaliero ← interesseAnnuo/**365**

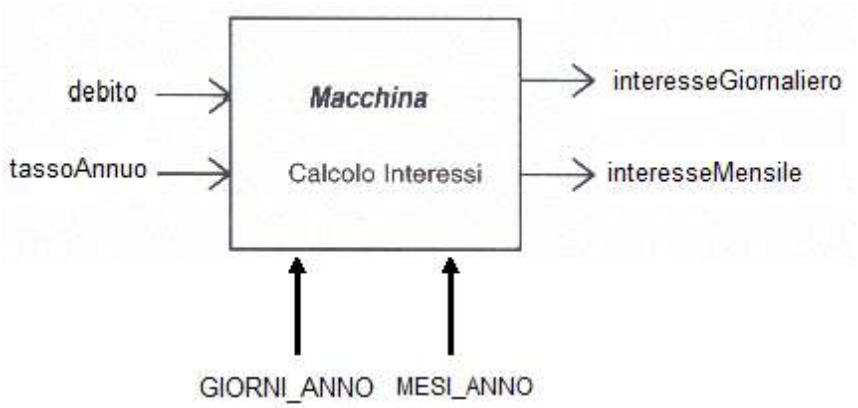
interesseMensile ← interesseAnnuo/**12**

visualizza interesseGiornaliero, interesseMensile

}

Dove il numero **365** che esprime il numero di giorni in un anno ed il numero **12** che esprime il numero di mesi in un anno sono quantità esplicite **costanti** non evidenziate nel modello del sistema e che non richiederanno di essere dichiarate nel *linguaggio di programmazione* usato per implementare l'algoritmo.

Per rendere più chiaro il procedimento, e permettere un controllo su tali valori, si potrebbe assegnare loro un **nome identificativo** ed analizzare tali dati come parametri costanti (***variabili immodificabili***) sintetizzando col seguente ***modello del sistema***:



parametri costanti

numeri interi GIORNI_ANNO = 365 *senza considerare anni bisestili*
MESI_ANNO = 12

In tal caso richiederanno di essere dichiarate nel *linguaggio di programmazione* usato per implementare l'algoritmo.

Esempio di **implementazione** dell'algoritmo come **applicazione** in linguaggio Java

(sviluppo in ***ambiente***[3] ***JCreator***)

```
/**
 * @(#)CalcoloInteressi.java
 *
 * CalcoloInteressi application
 *
 * @author Classe 3AI
 * @version 1.00 2013/12/02
 */

public class CalcoloInteressi  {
                                                    // metodo principale
        public static void main(String[] args) {
                                                    // scelta del tipo dei dati
            double debito, tassoAnnuo,
                   interesseAnnuo, interesseGiornaliero, interesseMensile ;

                                                    // costanti cioè variabili immodificabili
            final int  GIORNI_ANNO = 365,
                       MESI_ANNO = 12;

            // inizializzazione dei valori delle variabili: da programma in fase di test - senza lettura

            tassoAnnuo = 17.3 ;                     // espresso in percentuale: 17,3 %
            debito = 1000.50;                       // espresso in Euro

            interesseAnnuo = debito * tassoAnnuo/100;
            interesseGiornaliero = interesseAnnuo/ GIORNI_ANNO;
            interesseMensile = InteresseAnnuo/ MESI_ANNO;

            // visualizza su video in sezione General Output

            System.out.println("Interesse giornaliero " + interesseGiornaliero + " Euro");
            System.out.println("Interesse mensile " + interesseMensile + " Euro");

        }//fine metodo principale

    } // fine classe …. descrizione del risolutore di Calcolo di interessi
```

Effetto:

```
General Output
--------------------Configuration: CalcoloInteressi
Interesse giornaliero 0.4742095890410959 Euro
Interesse mensile 14.423875 Euro

Process completed.
```

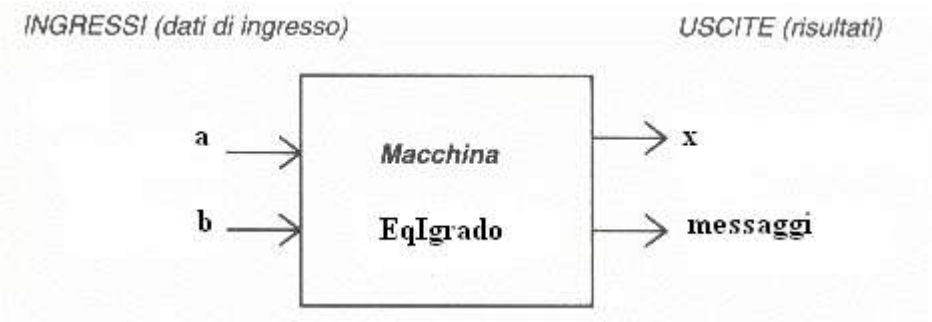
[3] L'ambiente di sviluppo (scaricabile gratuitamente da http://www.jcreator.com/download.htm selezionando JCreator **LE v4.50** oppure direttamente da http://xinox-jcreator-le.software.informer.com/4.5/) è un esempio di **I**ntegrated **D**evelopment **E**nvironment. Nel sito una guida introduttiva. Prima di installare tale IDE scaricare ed installare il kit di sviluppo JDK e la relativa documentazione (API)

Esempio:

Il problema è la risoluzione di un'equazione di primo grado, ridotta nella forma normale:

$$a*\mathbf{x} = b$$

modello del sistema



modello di dati

I dati sono **numeri reali**, *variabili* da memorizzare:
Input: i coefficienti **a** e **b** impostati da programma
Output: la soluzione **x** o messaggi informativi (*stringhe*) su video

modello matematico

x = b/a

Algoritmo in pseudocodice - modello logico

{
inizializza i coefficienti da programma

se (**a è diverso da zero**)

x ← b/a
stampa il valore di **x**

altrimenti se (**b è diverso da zero**)

visualizza messaggio "**Non esiste soluzione reale e finita**"
altrimenti
visualizza messaggio "**La soluzione è indeterminata**"

}

Segmento in **linguaggio di programmazione Java** per implementare il *costrutto* ***alternativa*** …. ***ternaria***

```
if ( a != 0) {

    x = b/a;                                  // operazione di assegnamento
    System.out.println("\n x = " + x);        // operatore + per concatenazione di stringhe
  }

else  if (b != 0)
                    System.out.println("\n Non esiste soluzione reale e finita\n\n") ;
      else
                    System.out.println("\n La soluzione e' indeterminata\n\n") ;
```

Glossario

Problema risolvibile *(dal calcolatore)*

qualsiasi quesito la cui *soluzione* possa essere rappresentata da una macchina (*automa: sistema discreto* cioè caratterizzato da un avanzamento per passi successivi dove ogni ***passo*** produce nell'automa un cambiamento di *stato;* inoltre ogni stato è descrivibile mediante una stringa finita di *simboli* tratti da un insieme prefissato) o, equivalentemente, da un *programma* per un calcolatore.

Capire se un problema è **calcolabile** effettivamente, significa capire se esso **ammette una soluzione algoritmica** (è risolvibile mediante calcolatore) **indipendentemente** da quanto **tempo/spazio** è necessario per risolverlo (*semialgoritmo*). Il termine calcolabilità distingue tra problemi che possono essere **definiti matematicamente** (*funzioni*) e quelli che invece ammettono una **soluzione effettiva o algoritmica**.

Un problema si dice di **complessità limitata** se è **risolvibile** (dal calcolatore) **avendo a disposizione una quantità limitata di risorse**, siano esse il tempo o la memoria (spazio).

L'**informatica come scienza**[4] si fonda proprio sulla **teoria della calcolabilità effettiva** sviluppata nell'ambito dello studio dei fondamenti della matematica, intorno agli anni '30.

Tale avventura intellettuale fornisce non solo le basi concettuali e teoriche dell'informatica, ma rappresenta anche un affascinante viaggio verso i limiti dell'informatica stessa.

Ogni disciplina scientifica si definisce pienamente nel momento in cui essa viene delimitata da una teoria in grado di evidenziarne i limiti e le potenzialità

E' così per la fisica classica e quantistica, per la psicoanalisi, per la chimica etc.

Anche l'informatica ha, come ogni scienza, una **teoria** che ne definisce in modo universale i limiti e le potenzialità.

Tale disciplina, la cui nascita si può far risalire agli anni '30 in un effervescente panorama culturale e scientifico di inizio secolo (sono di quel periodo gli studi sulla materia, sulla meccanica quantistica e sui fondamenti della matematica) si sviluppa pienamente indipendentemente dalla realizzazione, avvenuta solo successivamente negli anni '40, del primo calcolatore elettronico.

Possiamo quindi affermare, in modo forse provocatorio, che il successo attuale dell'informatica realizzata mediante dispositivi elettronici piuttosto che biomolecolari, quantistici o altro, sia stato più un caso che ha voluto la maturazione contemporanea dell'**informatica come scienza** e dell'elettronica intorno agli anni '40 e '50, piuttosto che una necessità intrinseca del processo di calcolo.

L'**indipendenza da una particolare macchina fisica** è uno dei punti di forza dell'informatica come scienza.

L'errore comunemente compiuto di identificare l'informatica con il computer moderno, o almeno con l'architettura che conosciamo oggi, ancora basata sulle idee di von Neumann-Turing, limita questa disciplina ad una mera programmazione di particolari macchine.

Al contrario, l'informatica prescinde dal particolare strumento di calcolo, sia esso il computer moderno, un insieme di molecole (DNA-computing) o particelle (Quantum-computing), o sia esso definito da un mero calcolo simbolico (*Lambda*-calcolo).

In questo senso l'informatica può definirsi a pieno titolo come una **scienza universale dell'informazione**, ovvero di come l'informazione possa essere codificata, manipolata, valutata, analizzata e misurata.

---

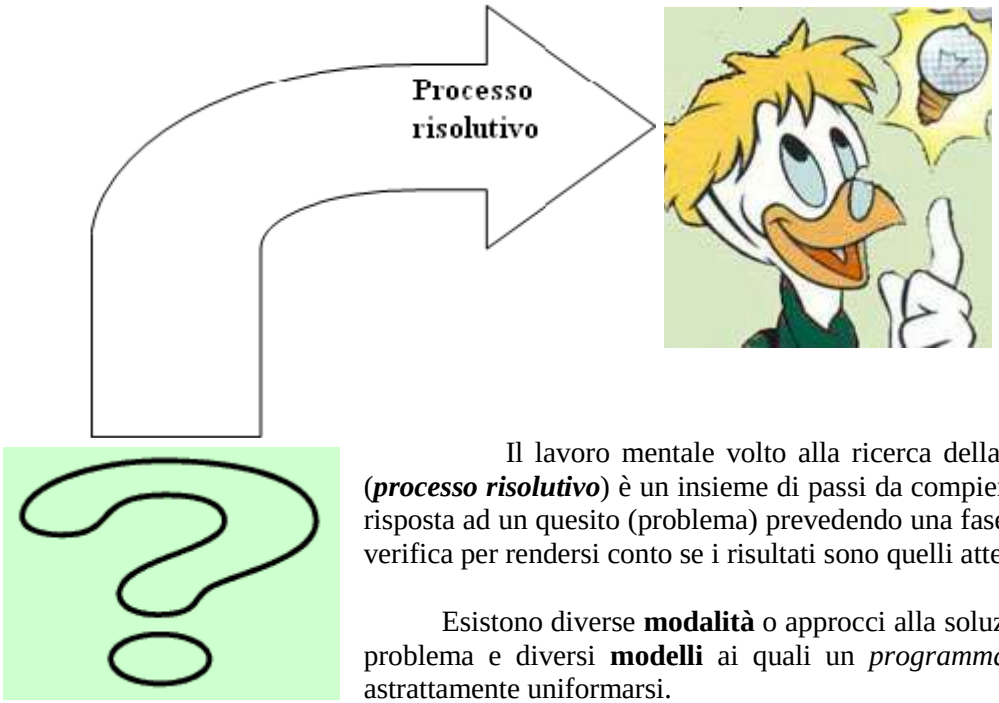
[4] Tratto da dispensa http://users.dimi.uniud.it/~agostino.dovier/DID/dispensa.pdf

Paradigma

E' un termine derivante dal greco *paradeigma* che significa «modello» (o «progetto») ed «esempio»; altra accezione del termine è prossima a quella di «analogia» (in Platone si trova usato in tutte le accezioni).

Le idee sono infatti modelli o termini di paragone assoluti, conoscendo i quali è possibile decidere se qualcosa sia o non sia conforme ad essi: per esempio conoscendo che cosa è la santità, si può giudicare di un'azione se sia o non sia santa. Col significato di «esempio» o «caso esemplare», Socrate si considera un *paradeigma* di cui il dio si è servito per illustrare la condizione umana riguardo al sapere, che è appunto di non-sapere. Aristotele assume *paradeigma* come termine tecnico della logica e della retorica col significato di «argomento fondato su un esempio» o «induzione retorica», dal particolare al particolare; è l'argomento che si trae da un caso noto per illustrare, grazie alla pregnanza dell'esempio che si è scelto, un caso meno noto o affatto ignoto. http://www.riflessioni.it/enciclopedia/paradigma.htm

Paradigma di programmazione: modello ("chiave di interpretazione", "punto di vista") cui la mente di chi legge o scrive un *programma* deve idealmente conformarsi



Il lavoro mentale volto alla ricerca della soluzione (***processo risolutivo***) è un insieme di passi da compiere per dare risposta ad un quesito (problema) prevedendo una fase di attenta verifica per rendersi conto se i risultati sono quelli attesi.

Esistono diverse **modalità** o approcci alla soluzione di un problema e diversi **modelli** ai quali un *programmatore* deve astrattamente uniformarsi.

A seconda della **metodologia** si definiscono[5] **diversi paradigmi di programmazione:**

- **procedurale** se affrontando i problemi in **modo tradizionale** si seguono le tre fasi seguenti:
  - **Conoscenza** e studio della *realtà* (o contesto) da cui è tratto il problema
  - **Comprensione** del problema ("**cosa**" si vuole)
  - **Ricerca** di una strategia (o idea) risolutiva per il problema ("**come**") e sintesi della procedura risolutiva (*algoritmo* e/o programma) per un esecutore

[5] Si vedranno in seguito altri paradigmi di programmazione: non procedurale (dichiarativo), *event-driven* etc…

Linguaggio

Il **linguaggio** consente la **comunicazione**, intesa come scambio di informazioni (elementi di conoscenza significativa).

Un'**informazione** può:

- essere acquisita tramite rilevamento diretto di un evento reale, mediante percezione sensoriale;
- essere trasmessa intenzionalmente (*messaggio*) da un emittente al ricevente per mezzo di un canale.

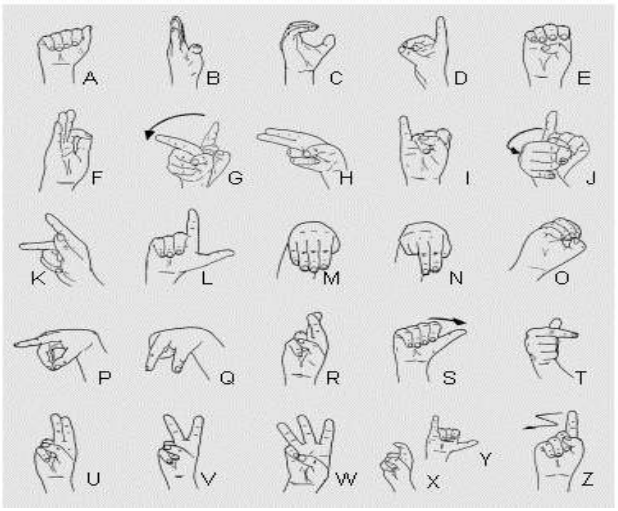
Nel secondo caso la sorgente dell'informazione sceglie la rappresentazione più opportuna dell'informazione, affinché il messaggio sia compreso in modo univoco ed obiettivo. L'*emittente* è colui dal quale parte l'informazione, il mezzo che trasporta l'informazione è detto canale. Il ricevente è il destinatario dell'informazione.

*La funzione fondamentale del linguaggio è quella di sostituire ad oggetti o concetti dei simboli. Ogni parola (**significante**) rappresenta un oggetto concreto, una persona, un ente astratto (**significato**).*

Il linguaggio è un *sistema di segni*, sistema in quanto i vari elementi si cui è formato funzionano insieme in un tutto unitario. I segni sono le parole (*stringhe*) del linguaggio, che si usano per indicare l'associazione tra un dato percepibile ed un concetto. Ogni parola (*significante*) rappresenta un oggetto concreto, una persona, un ente astratto che la mente umana associa al messaggio (*significato*).

Elementi di un linguaggio

•**alfabeto**: insieme finito e non vuoto di ***simboli*** convenzionali espressi con segni tipografici detti caratteri



•**sintassi**: insieme finito e non vuoto delle ***regole*** mediante le quali si formano le *stringhe* o le frasi di un linguaggio

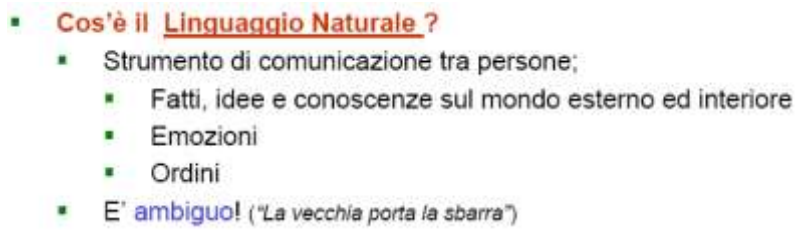
•**semantica**: insieme finito e non vuoto di ***significati*** da attribuire alle stringhe

•**grammatica**: insieme finito e non vuoto di tutte le ***regole*** che servono per generare un *linguaggio*

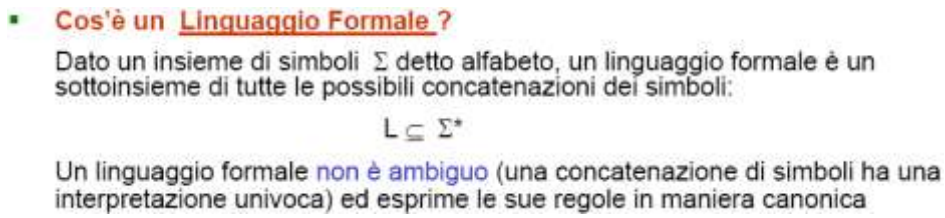
Linguaggi naturali e formali

Distinguiamo tra:

- **linguaggi naturali**, utilizzati nella comunicazione quotidiana tra gli esseri umani, privi di una definizione rigorosa ed in continua evoluzione, spesso presentano ambiguità ma hanno enorme potenza espressiva;



- **linguaggi formali**, creati dall'uomo per scopi precisi, secondo **regole** convenzionali esplicite che non ammettono eccezioni e non consentono sinonimi e omonimie. Il significato di una frase di tali linguaggi è sempre privo di ambiguità, è sempre possibile determinarne la correttezza (grammaticale), ma essi hanno potere espressivo limitato.



La funzione del linguaggio formale nella comunicazione uomo-macchina

L'utilizzo di un linguaggio formale consente di passare da un algoritmo al corrispondente programma, **codificando** opportunamente le istruzioni

Un elaboratore può riconoscere e generare solo Linguaggi Formali, attraverso l'utilizzo di modelli e algoritmi

Una prima distinzione può essere fatta tra:

1. **linguaggi non evoluti o di basso livello** (linguaggio macchina, linguaggi assemblativi);
2. **linguaggi evoluti o di alto livello** (algoritmici, speciali).

Ogni processore ha un proprio linguaggio, detto **linguaggio macchina,** che ad ogni stringa di bit fa corrispondere una operazione elementare. Questo tipo di linguaggio è molto lontano dal modo di ragionare dell'uomo. Inizialmente l'attività di codifica di algoritmi, lavoro lungo e difficile, era riservata solo a tecnici super specializzati. In seguito furono creati dei linguaggi intermedi con cui scrivere i programmi. Per l'utilizzo di un algoritmo codificato in questo modo è necessario un apposito programma traduttore che converte il programma originale (detto *file sorgente*) nelle corrispondenti istruzioni in linguaggio macchina (ottenendo così il file oggetto). Il primo di tali linguaggi fu il **linguaggio Assembler**, in cui al posto di ogni istruzione macchina viene usato un *codice mnemonico* ad esso associato. L'Assembler è ancora molto vicino alla logica del processore.

Successivamente sono stati ideati linguaggi non più orientati alla macchina, ma **orientati alla soluzione di problemi**. Con tali linguaggi, indipendenti dal processore e dalla particolare macchina, il lavoro del programmatore risulta semplificato. Le istruzioni e i dati da esse manipolati vengono *rappresentati simbolicamente* e viene delegato alla macchina il lavoro ripetitivo e deterministico della loro traduzione in codifiche binarie.

Categorie dei linguaggi evoluti

Sono elencate di seguito le principali categorie[6] in cui si classificano i linguaggi di alto livello.

Linguaggi imperativi: il programma è costituito da una sequenza di istruzioni che modificano il contenuto della memoria dell'elaboratore o determinano le modalità di esecuzione di altre istruzioni; assume un ruolo fondamentale l'*istruzione di assegnazione*. Sono imperativi la maggior parte dei linguaggi più diffusi (Pascal, Basic, Fortran, C, Cobol, ecc.).

Linguaggi funzionali: il programma è considerato come il calcolo del valore di una funzione; in un linguaggio funzionale *puro* l'assegnazione esplicita è completamente assente e si utilizza solo il passaggio dei parametri. Rivestono particolare importanza la *ricorsione*, ossia l'utilizzo di funzioni che richiamano se stesse e, come struttura dati, la lista (sequenza ordinata di elementi). Il più importante rappresentante di questa categoria è il Lisp (LISt Processing).

Linguaggi dichiarativi: il programma è considerato come la dimostrazione della verità di una proposizione; il sorgente è costituito da una sequenza di asserzioni di fatti e regole. Non è indicato esplicitamente il flusso di esecuzione, ma dato un obiettivo di partenza (*il goal*) è il sistema che cerca di individuare i fatti e le regole rilevanti. La parte dichiarativa (il *cosa fare*) e la parte procedurale (il *come*) sono nettamente separate, favorendo la leggibilità del programma.I linguaggi logici risultano adatti a risolvere problemi che riguardano *entità* e le loro *relazioni*. Più in generale si parla di **paradigma non procedurale** che prevede di specificare la conoscenza della *realtà* e il problema in un *linguaggio di specifica* ed affidare all'interprete del linguaggio la soluzione.

Linguaggi strutturati: la programmazione strutturata ha lo scopo di semplificare la struttura dei programmi, limitando l'uso delle strutture di controllo a pochi casi semplici. L'uso del salto incondizionato (GOTO) viene sostituito da istruzioni di controllo del flusso più strutturate (WHILE, FOR,UNTIL)

Linguaggi orientati ad oggetti: il programma è considerato l'effetto dell'interazione di un insieme di oggetti (insiemi di dati e algoritmi che manipolano questi dati) che comunicano con l'esterno mediante messaggi. Oltre a linguaggi specializzati che implementano i principi di tale metodologia (Smalltalk, il linguaggio Java), sono nate delle estensioni dei linguaggi già esistenti, che li integrano (ad es. C++ per il C, CLOS per il Lisp). Più in generale si parla di **paradigma OO**.

Linguaggi orientati agli eventi: in un ambiente *event driven* non esiste più una sequenza determinata di comandi da eseguire ma una serie di reazioni che il sistema ha. rispondendo a determinati stimoli esterni o interni.Più in generale si parla di **paradigma *event driven***.

Linguaggi di Programmazione Visuale (**Visual Programming Language V.P.L.**): linguaggi che consentono la programmazione tramite la manipolazione grafica degli elementi e non tramite sintassi scritta. Un VPL consente di programmare con "espressioni visuali" ma anche, all'evenienza, di inserire spezzoni di codice (solitamente questa funzione è riservata a formule matematiche). La maggioranza dei VPL è basata sull'idea "boxes and arrows" ovvero i box (o i rettangoli le circonferenze ecc...) sono concepiti come funzioni connesse tra di loro da "Arrows" le frecce.

I VPL possono essere ulteriormente classificati, a seconda di come rappresentano su schermo le funzioni, in i*con-based*, *form-based* , o linguaggio *a diagrammi*. L'ambiente per la programmazione visuale provvede tutto il necessario per poter "***disegnare***" sin da subito un programma; in rapporto ai linguaggi scritti le regole sintattiche sono praticamente inesistenti. Un esempio: il linguaggio G usato in ambiente LabVIEW

I vantaggi della programmazione visuale sono incredibili, oltre ad una facilità di apprendimento e alla capacità di poter "vedere il programma" durante le fasi debug, la programmazione parallela (se gestita dal software) diviene quasi "istintiva" e soprattutto eseguita in automatico.

[6] Si consultino nel sito la classificazione dei linguaggi di programmazione all'interno della classificazione del SW e l'estratto da Ricky Spelta

Elaborazione del linguaggio naturale ed IA

Il Linguaggio Naturale può essere rappresentato attraverso un Linguaggio Formale ?

Richard Montague (matematico e filosofo americano) è stato uno dei primi nella sua semantica modellistica a tentare un approccio al linguaggio naturale con gli strumenti della logica (i suoi lavori in quest'area risalgono alla fine degli anni Sessanta inizi Settanta)

Nel suo articolo dal titolo provocatorio *English as a Formal Language*, scrisse "Rigetto l'affermazione che vi sia una sostanziale differenza teorica tra linguaggi formali e naturali". In questo articolo, e ancora più sistematicamente in *Montague 1970*, diede, quindi, un modo per descrivere la sintassi e la semantica (e loro relazione) dei linguaggi naturali in una maniera simile a quanto usato per i linguaggi formali.

E' solo un sogno[7] l'intelligenza artificiale?

- **Hal :** - *Buonasera, David.*
- **David:** - Come va, Hal?
- **Hal :** - *Va tutto benissimo, e tu?*
- **David:** - Beh, non c'è male.
- **Hal :** - *Hai fatto dell'altro lavoro?*
- **David:** - Si', qualche disegno.
- **Hal :** - *Posso vederlo?*
- **David:** - Certo.
- **Hal :** - *Un'ottima esecuzione, David. Mi pare che tu abbia migliorato parecchio.*

(da 2001: Odissea nello Spazio)

HAL
-Ascolta
-Analizza
-Comprende
-Ragiona
-E' pragmatico
-Risponde

Chi (o cosa) è Hal ?
Una intelligenza "Artificiale" o "Naturale" ?

- *Due punti di vista sull' Intelligenza Artificiale:*
- **IA forte** — *"Ragionare è calcolare", Thomas Hobbes*
  - il calcolatore può essere dotato di intelligenza pura, non distinguibile da quella umana, può essere cosciente di se
- **IA debole** : — *"Nessun programma, da solo, è sufficiente a ragionare intenzionalmente", John R. Searle*
  - il calcolatore ha un'intelligenza parziale, non comparabile a quella umana, può solo emulare i processi cognitivi umani

NPL:

L' Elaborazione del Linguaggio Naturale (*Natural Language Processing, NLP*) si prefigge come obiettivi principali:

- costruzione di modelli e e strumenti informatici in grado di eseguire specifici task riguardanti il Linguaggio Naturale:
  - Permettere la comunicazione *uomo – macchina*
  - Migliorare la comunicazione *uomo – uomo*
  - Elaborare e manipolare oggetti linguistici

[7] Il film **AI** (tratto da un romanzo di Brian Aldiss: **Supertoys Last All Summer Long**) e VR al cinema

PERCHE' E' IMPORTANTE L' NLP ?

- Sempre maggiore quantità di conoscenza condivisa in testi in Linguaggio Naturale *machine readable (ES: il Web)*
- Necessità di un'interazione più diretta uomo-macchina (*ES: agenti intelligenti)*

Esempi

Google Translate

This text has been automatically translated from Italian:

Hal: - Buonasera, David. David: - As he goes, Hal? Hal: - He goes very well all, and you? David: - Beh, not is badly. Hal: - you have made of the other job? David: - ', some design. Hal: - I can see

MACHINE TRANSLATION: tradurre automaticamente testi in lingue diverse

(*translate.google.com/translate_t*)

Hal : - *Buonasera, David.*	-*Buonasera, David.*
David: - Come va, Hal?	-As he goes, Hal?
Hal : - *Va tutto benissimo, e tu?*	*- He goes very well all, and you?*
David: - Beh, non c'è male.	- Beh, not is badly.
Hal : - *Hai fatto de*	*f the other job?*
David: - Si', qualc	
Hal : - *Posso vederlo?*	*- I can see it?*
David: - Certo.	- Sure.
Hal : - *Un'ottima esecuzione, David. Mi pare che tu abbia migliorato parecchio.*	*- an optimal execution, David. It seems to me that you have improved much.*

My name is Marco → اسمي ماركو

Esempi

Cyber Letteratura

UNIVERSITÀ DEGLI STUDI DI ROMA Tor Vergata Università

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

ANALISI NARRATIVA: aiutare un critico ad analizzare un testo letterario

(*ai-nlp.info.uniroma2.it/cyberletteratura/*)

RICERCA NARRATIVA

Tag Narrativo descrizione luogo esterno oggettiva analitico

Sub-Tag Narrativo azione punto di vista Carla

Cerca

Da *"Gli Indifferenti"* di Alberto Moravia

1.[Capitolo VIII] ma ad una certa distanza tutto si confondeva, gli alberi si torcevano come serpenti, tutto si annebbiava;

2.[Capitolo VIII] Alzò gli occhi, e vide davanti a sé la macchia nera del cancello, i due pilastri bianchi, il fogliame scuro di un grande albero curvo sotto la pioggia; aprì la portiera di servizio, uscì sulla strada volgendo gli occhi alla parte opposta a quella dove Leo aspettava

3.[Capitolo XVI] Carla seguiva attentamente la corsa e con quella stessa velocità i pensieri turbinavano nella sua mente occupata e stanca. l'automobile era la sua vita, lanciata ciecamente nell'oscurità

La "macchina" universale di Turing

il test di Turing *in lingua originale* http://loebner.net/Prizef/TuringArticle.html
in italiano http://it.wikipedia.org/wiki/Test_di_Turing
riformulato http://www.nemesi.net/turingtest.htm

Alan Turing, in un articolo del 1950 *Computing Machinery and Intelligence* (Macchine calcolatrici e intelligenza), propose un criterio - oggi noto come "test di Turing" - per determinare se un computer fosse in grado di pensare. Turing era convinto che la sua macchina potesse effettuare qualsiasi operazione logica e, programmata con la necessaria abilità, entro il duemila avrebbe potuto simulare l'intelligenza umana.

Ecco come si dovvrebbe svolgere il test di controllo. Una persona si trova davanti ad un terminale e con la tastiera scrive delle domande e riceve delle risposte. Dall'altro capo del terminale ci sono una macchina ed un operatore umano che forniscono alternativamente le risposte alle domande. Se la persona non è in grado di distinguere quando sta interloquendo con una macchina e quando con un operatore umano, allora la macchina è intelligente.

Finora nessun programma ha superato il test di Turing. Il primo ad aver ottenuto un certo successo, è *Eliza*, un programma scritto nel 1966 da Joseph Weizenbaum. Eliza è una psicoterapeuta che simula una conversazione tra lei (il medico), e voi (il paziente).

Il test consiste in un gioco, noto come ***gioco dell'imitazione***, a tre partecipanti: un uomo A, una donna B, e una terza persona C.

Quest'ultimo è tenuto separato dagli altri due e tramite una serie di domande deve stabilire qual è l'uomo e quale la donna.

Dal canto loro anche A e B hanno dei compiti: A deve ingannare C e portarlo a fare un'identificazione errata, mentre B deve aiutarlo.

Affinché C non possa disporre di alcun indizio (come l'analisi della calligrafia o della voce), le risposte alle domande di C devono essere dattiloscritte o similarmente trasmesse.

Il test di Turing si basa sul presupposto che una macchina si sostituisca ad A. In tal caso, se C non si accorgesse di nulla, la macchina dovrebbe essere considerata intelligente, dal momento che - in questa situazione - sarebbe indistinguibile da un essere umano.

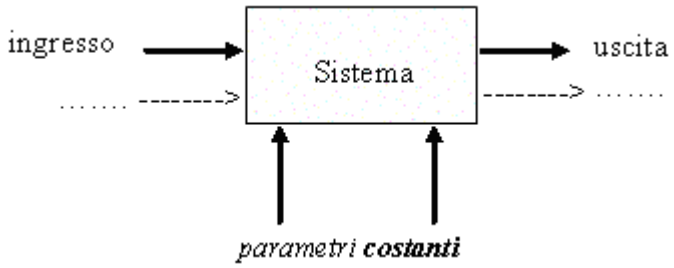
Alan Mathison Turing

Foto da “Biografie dei padri dei computer”

http://www.windoweb.it/edpstory_new/ep_turing.htm

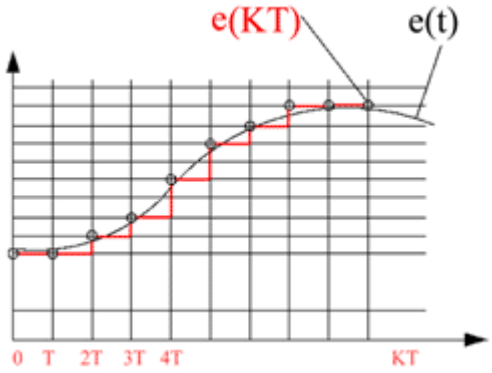
Rappresentazione *sistemica* (paradigma ingresso-uscita): descrizione a blocchi funzionali

Rappresentazione grafica che distingue tra variabili in ingresso *(grandezze su cui possiamo agire per introdurre modifiche)* e in uscita o risposte *(grandezze che risultano influenzate e possiamo osservare per studiare sperimentalmente l'andamento)* individuando gli eventuali *parametri costanti* :



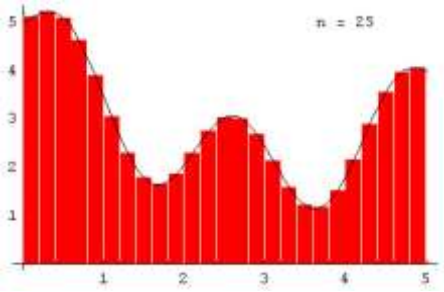
nb: Nel caso di simulazione al computer, uno dei *parametri* ***costanti*** è il tempo di discretizzazione **Δt** cioè l'intervallo tra campioni da fissare poiché un **esecutore discreto** è in grado di distinguere solo un **numero finito di valori** e non consente di analizzare infiniti valori istantanei.

DISCRETIZZAZIONE DI UN SEGNALE ANALOGICO: la discretizzazione è uno dei due processi che costituiscono la **digitalizzazione**. La discretizzazione può essere chiamata anche "quantizzazione" poichè sostituisce ai valori *continui* del segnale analogico dei valori *discreti* (quantificati, nel senso letterale di "quanto") cioè non-continui, "a salti. La figura qui sotto chiarisce quanto detto; **e(t)** rappresenta il segnale analogico continuo, mentre **e(KT)** quello discretizzato (o quantificato).

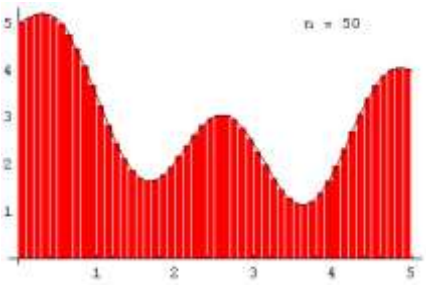


I circoletti indicano il valore che assume il segnale dopo la **discretizzazione**; collegandoli, supponendo noto il segnale esclusivamente negli istanti di tempo campionati, si ottiene un andamento "a gradini" rappresentativo del segnale discretizzato e campionato. Sull'asse verticale sono segnati i possibili salti (livelli) di valore che quantificano il segnale; più essi sono fitti, maggiore è la precisione della discretizzazione.

Caso numero campioni pari a 25



Caso numero campioni pari al doppio (scelta del tempo di discretizzazione **Δt** pari alla metà) : maggiore precisione



Tanto **minore è Δt**, tanto maggiore è la precisione

Estratti dai testi: G. Callegarin "Nuovo corso di informatica - ABACUS" CEDAM
M. Romagnoli, P. Ventura "Linguaggio C e C++" PETRINI
D. Fuselli, A. Henin, G. Vallini "Sistemi continui e discreti" ZANICHELLI