

Fondamenti di Informatica

Prof. Ing. Loris Penserini, PhD

elpense@gmail.com

<https://orcid.org/0009-0008-6157-0396>

Materiale:

<https://github.com/penserini/>

Tecnologie Adottate nel Corso

Strumenti utilizzati in questo modulo per realizzare algoritmi:

- **Flowgorithm**: esegue progetti scritti in Flow-Chart, e consente anche di tradurre in altri linguaggi: pseudocodice, C++, PHP, Python, ecc.
Per Windows: <http://www.flowgorithm.org/download/>
- **PYTHON e Visual Studio Code come IDE di Sviluppo principale**
- **IoT**: domotica smart con dispositivi Shelly e Tapo
- **LLM e Trasformer**: creare un assistente AI basato su Ollama

Altre tecnologie che verranno usate meno:

- **Eclipse**: *Integrated Development Environment* per sviluppare software in diversi linguaggi. Si è utilizzata la versione per Java.
<https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers>
- **Netbeans + XAMPP** : IDE per alcuni esempi di OOP in PHP
- **JDK – Oracle**: strumenti di sviluppo per Java.
<https://www.oracle.com/java/technologies/downloads/#jdk18-windows>

Scienza Informatica

L'informatica è la scienza che studia i principi e le tecniche per rappresentare, elaborare e trasmettere l'informazione. Essa comprende sia la modellazione del mondo reale tramite strumenti formali e computazionali, sia la progettazione di sistemi e procedure capaci di elaborare informazione in modo autonomo.

Attraverso specifici linguaggi di programmazione, si possono scrivere algoritmi (sequenze di istruzioni) che un calcolatore elettronico è in grado di elaborare in modo automatico al fine di fornire delle informazioni utili all'utente partendo da dati iniziali.

Ciascun paradigma di programmazione fornisce allo sviluppatore di software un differente approccio concettuale per implementare il pensiero computazionale, cioè la definizione della strategia algoritmica utile per digitalizzare e **risolvere un problema del mondo reale**.

L'evoluzione delle tecnologie digitali nei settori della Robotica e dell'AI, spingono i ricercatori a sviluppare nuovi paradigmi di programmazione...

Scienza Informatica e Mondo Reale

Storicamente e pragmaticamente, anche gli aspetti più teorici dell'informatica (come i limiti del calcolo, la complessità o la teoria degli automi) hanno avuto senso perché collegati all'esigenza di modellare e automatizzare compiti del mondo reale.

La complessità computazionale: serve a capire quali problemi del mondo reale sono trattabili in pratica (es. pianificazione, ottimizzazione, crittografia) e quali invece rimangono troppo costosi.

La teoria degli automi e dei linguaggi formali: nasce per modellare linguaggi naturali o di programmazione, cioè modi per descrivere compiti reali da affidare a una macchina.

I limiti del calcolo (problema della fermata, indecidibilità, ecc.): hanno un impatto diretto sulla consapevolezza di quali aspetti del mondo reale possono essere automatizzati e quali resteranno sempre fuori portata.

Scienza Informatica è al servizio dell'uomo

Immagina l'automobile: Prima di essa, l'uomo percorreva distanze a piedi o a cavallo. L'auto non ha inventato il viaggio: lo ha reso più rapido, efficiente, accessibile. Per costruirla, è stato necessario fondere insieme meccanica, termodinamica, elettronica, materiali: discipline diverse, tutte finalizzate a un unico scopo concreto — muoversi dal punto A al punto B in minor tempo.


Così è l'informatica: Non nasce dal nulla, ma dall'esigenza di rappresentare e trattare informazione per raggiungere obiettivi del mondo reale. Anche qui confluiscano logica, matematica, elettronica, linguistica, intelligenza artificiale: saperi diversi che cooperano per un fine comune — automatizzare o ampliare compiti tipicamente umani.

L'informatica, quindi, non è un gioco teorico fine a se stesso. È il motore che, modellando il mondo reale, ci permette di percorrerlo in modi nuovi, più veloci e più potenti, proprio come l'automobile ha trasformato il nostro modo di muoverci nello spazio.



Allora ecco una definizione più consona...

L'informatica è al sapere ciò che l'automobile è al movimento: integra discipline diverse per modellare il mondo reale e trasformare compiti umani in procedure automatiche, riducendo tempi e limiti delle nostre capacità, proprio come l'auto ha reso più rapido ed efficiente lo spostamento da un luogo all'altro.



Programmazione

La maggior parte dei linguaggi di programmazione si basano sul «**paradigma imperativo**», in cui il programma consiste in un insieme di istruzioni dette anche «**direttive**» o «**comandi**».

La programmazione imperativa viene generalmente contrapposta a quella dichiarativa, in cui un programma consiste in un insieme di «affermazioni» (non «comandi») che l'esecutore è tenuto a considerare vere e/o rendere vere. Un esempio di **paradigma dichiarativo** è la programmazione logica (es. Prolog, LTL, CTL, ecc.).

Paradigmi

➤ Storicamente i principali Paradigmi di programmazione

- **Imperativo**

- Il programma è una sequenza di istruzioni che modificano lo stato.
- Esempi: C, Fortran, Assembly.

- **Procedurale**

- Variante dell'imperativo, organizza il codice in procedure o funzioni.
- Esempi: C, Pascal.

- **Dichiarativo**

- Si specifica **cosa** deve essere calcolato, non **come**.
- Esempi: SQL, Prolog.

La contestazione dello «spaghetti code»

Negli anni '60 e '70, lo sviluppo software cresceva in complessità.

I programmi erano spesso scritti in Fortran, COBOL o in Assembly, e facevano ampio uso del comando **goto**, che saltava arbitrariamente da un punto all'altro del codice.

Il risultato erano programmi difficili da leggere, mantenere e verificare: il cosiddetto “**spaghetti code**”, per l'intreccio caotico del flusso di controllo.

➤ 1965 – Corrado Böhm e Giuseppe Jacopini

Pubblicano il teorema **Böhm–Jacopini**, dimostrando che ogni algoritmo può essere espresso usando solo tre strutture di controllo:

- **sequenza**,
- **selezione** (if/then/else),
- **iterazione** (while/for).

Questo fu il fondamento teorico della **programmazione strutturata**.

Il Periodo Storico

1968 – Edsger W. Dijkstra

Scrive la famosa lettera “Go To Statement Considered Harmful” pubblicata su Communications of the ACM. Denuncia l'abuso del goto come causa di software caotico e sostiene l'approccio strutturato.

Anni '70 – Niklaus Wirth

Porta avanti questi principi nello sviluppo dei linguaggi **Pascal** e **Modula**, che promuovevano l'uso disciplinato delle strutture di controllo e la modularità.

Donald Knuth (anni '70)

Pur riconoscendo la validità delle critiche, sostenne che il goto poteva avere usi limitati e giustificati. La sua posizione fu più pragmatica, ma non ostacolò l'affermazione della **programmazione strutturata**.

Programmazione Strutturata

Questa “contestazione” portò a:

- La nascita della programmazione strutturata come disciplina dentro il paradigma imperativo.
- La progettazione di linguaggi che integravano fin dall'inizio strutture di controllo ben definite, riducendo o eliminando il **goto** (es. Java).

La programmazione strutturata preparò il terreno ai paradigmi successivi, come il procedurale e l'orientato agli oggetti. Come vedremo in seguito, i concetti introdotti dalla programmazione strutturata sono alla base di numerosi altri linguaggi di programmazione, non ultimo quello orientato agli oggetti.

Cos'è la programmazione strutturata?

È un sotto paradigma dell'imperativo, nato negli anni '60–'70 come reazione al cosiddetto “spaghetti code” pieno di goto.

L'idea centrale del [teorema Böhm–Jacopini](#): qualsiasi programma può essere costruito combinando tre strutture di controllo:

- **Sequenza (esegui istruzioni in ordine)**
- **Selezione (if / else)**
- **Iterazione (while, for)**

A volte si aggiunge anche la composizione modulare (suddividere il programma in funzioni o procedure).

Esempio: prog. Strutturata vs Spaghetti code

Imperativo con «GOTO» (spaghetti-code):

```
c

int i = 0;
inizio:
if (i < 10) {
    printf("%d\n", i);
    i++;
    goto inizio; // tipico spaghetti code
}
```

Strutturato con ciclo:

```
c

for (int i = 0; i < 10; i++) {
    printf("%d\n", i);
}
```

Perché stiamo usando il linguaggio C e non Java o Python per gli esempi?

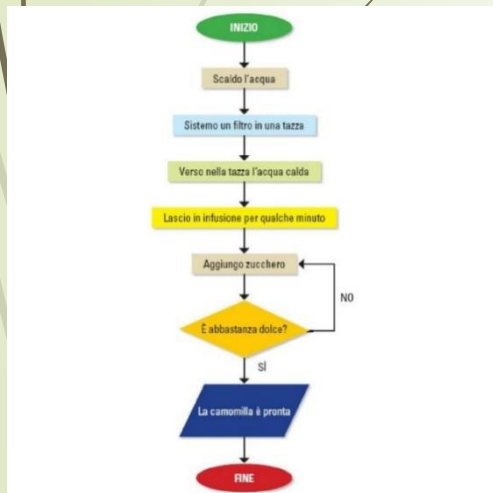
Paradigmi più diffusi oggi

- **Object-Oriented (OO, orientato agli oggetti)**
 - Basato su oggetti con stato e metodi.
 - Esempi: Java, C++, Python (in parte).
- **Funzionale**
 - Computazione come valutazione di funzioni pure, senza stato mutabile.
 - Esempi: Haskell, Lisp, Scala (in parte).
- **Logico**
 - Basato su regole logiche e inferenza automatica.
 - Esempi: Prolog, Datalog.

Comunicare con il Calcolatore

Indipendentemente dall'hardware, dal S.O. e dai linguaggi di programmazione, queste sono le fasi (**astrazioni**) necessarie per comunicare con un elaboratore.

PROGETTARE L'ALGORITMO



CODIFICARE L'ALGORITMO



COMPILARE L'ALGORITMO



ESEGUIRE L'ALGORITMO



ASTRAZIONE

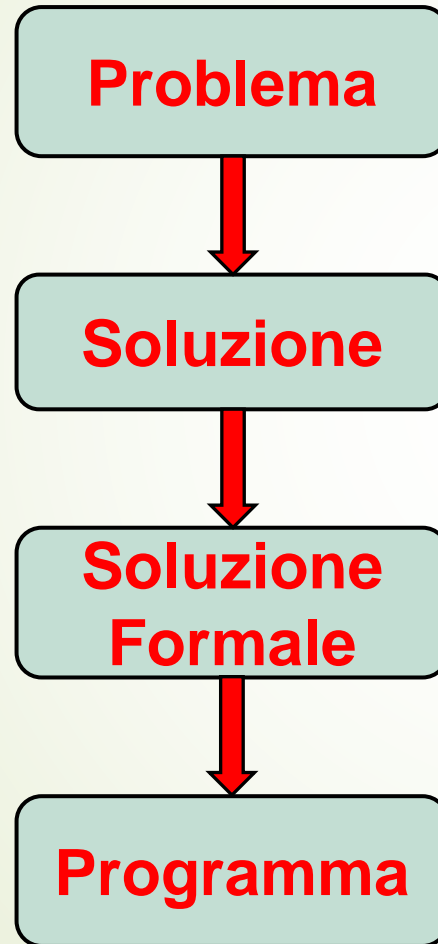
Scrivere un Algoritmo

A **design-time** un problema per essere risolto da un «elaboratore elettronico» o «esecutore» deve essere rappresentato con una formalizzazione che chiamiamo **ALGORITMO**, cioè nella logica del linguaggio di progetto con la quale si descrive il modello informatico di un problema del mondo reale.

L'algoritmo è definito da un insieme finito di passi (operazioni) che portano a risolvere un problema informatico, dove si hanno dei dati in ingresso (input) e si producono dei risultati (output).

La soluzione ad un problema del mondo reale può avere differenti algoritmi risolutori, tuttavia la programmazione cerca di trovare quello più efficiente.

Dall'Algoritmo al Programma



Esperienza da cui si
ricava l'idea

Metodi formali e strumenti automatici
per scrivere l'Algoritmo

Paradigmi di programmazione e
relativi linguaggi

Algoritmo e Programma

Algoritmo: una descrizione progettuale della logica in termini di sequenza di operazioni/istruzioni che descrive la soluzione al problema

Programma: il linguaggio formale con cui si esprime la logica dell'Algoritmo

Trovata la soluzione migliore al problema, questa viene formalizzata nella logica di progetto, ovvero la logica dell'**algoritmo**, che è composto da una sequenza finita di istruzioni e strutture di controllo. Questa logica algoritmica è vicina a quella che comprende l'«**esecutore automatico**» ovvero l'elaboratore elettronico.

Infine, l'algoritmo deve essere tradotto in un linguaggio formale di alto livello: il **programma**, al quale si può applicare il processo di **compilazione**.

Proprietà di un Algoritmo

Algoritmo: una sequenza finita e ben definita di passi logici che risolvono un problema.

Caratteristiche o proprietà:

- **Astratto:** cioè non dipende da un linguaggio di programmazione.
- **Indipendente dalla macchina:** può essere espresso in linguaggio naturale, pseudocodice, diagramma di flusso, UML, ecc.
- **Osservabile:** ogni operazione deve produrre, se eseguita, un effetto che possa essere descritto.
- **Deterministico :** Ogni operazione deve produrre lo stesso effetto ogni volta che viene eseguita a partire dagli stessi input e dalle stesse condizioni iniziali.
- **Corretto:** dare la soluzione giusta.
- **Finito:** terminare.
- **Efficiente:** ragionevole in tempo/risorse.

Programma

- **Definizione:** la traduzione concreta di un algoritmo in un **linguaggio di programmazione** (C, Python, Java, ecc.) che può essere **eseguito da un elaboratore in modo automatico**.
- **Caratteristiche:**
 - **Formale:** scritto con una sintassi precisa e regole del linguaggio scelto.
 - **Vincolato a un ambiente:** linguaggio, compilatore/interprete, sistema operativo.
 - **Non solo logica:** include anche dettagli pratici (gestione input/output, allocazione memoria, interfaccia utente...).

Esempio: Algoritmo di Euclide per il MCD

L'algoritmo di Euclide è un metodo efficiente per trovare il Massimo Comun Divisore (MCD) di due numeri interi, basato sulla proprietà che il MCD di due numeri è uguale al MCD di uno dei numeri e del loro resto, fino ad arrivare a un resto nullo; l'ultimo resto non nullo è il MCD cercato. Si esegue tramite divisioni successive, invece che sottrazioni, per velocizzare il processo, finché il resto non diventa zero.

Come funziona (Metodo delle Divisioni)

1. Dividi il numero maggiore per il numero minore e trova il resto: la formula è: $a = b * q + r$
2. Se il resto (r) è 0, il numero minore (b) è il MCD
3. Se il resto non è 0, sostituisci a con b e b con r , e **ripeti** il processo di divisione
4. Continua finché non ottieni un resto nullo
5. L'ultimo resto non nullo è il MCD.

Project Work: Applicazione di un Algoritmo

Utilizzando l'algoritmo precedente, calcolate il MCD di 96 e 36:
risolvere il problema ragionando in termini algoritmici.



GRAZIE!