

# Linguaggi di Programmazione I – Lezione 1

Prof. Marco Faella

<http://wpage.unina.it/mfaella>

Materiale didattico elaborato con i Proff. Sette e Bonatti

13 marzo 2024



# Panoramica della lezione

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Paradigmi computazionali



[Introduzione al corso](#)

[Obiettivi specifici](#)

[Obiettivi generali](#)

[Scheduling](#)

[Altre informazioni](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)

# Introduzione al corso



# Obiettivi specifici

[Introduzione al corso](#)

**Obiettivi specifici**

[Obiettivi generali](#)

[Scheduling](#)

[Altre informazioni](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)

- Mettere lo studente in grado di imparare velocemente un nuovo linguaggio di programmazione
  - ◆ mediante astrazione delle *caratteristiche costituenti* dei linguaggi
  - ◆ cambiano più di rado di quanto vengano introdotti nuovi linguaggi
- Spiegare le differenze tra i vari paradigmi di programmazione - imperativo, ad oggetti, funzionale, logico - e sul loro impatto sullo stile di soluzione dei problemi.
- Mostrare cenni sui criteri di progetto e le tecniche d'implementazione dei linguaggi di programmazione
- Esempi focalizzati su Java, ML, Prolog. Ma anche C, Pascal, e Python.



# Obiettivi generali

- Migliorare l'abilità nel risolvere i problemi usando meglio i linguaggi di programmazione.

[Introduzione al corso](#)

[Obiettivi specifici](#)

**Obiettivi generali**

[Scheduling](#)

[Altre informazioni](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)



# Obiettivi generali

- Migliorare l'abilità nel risolvere i problemi usando meglio i linguaggi di programmazione.
- Imparare a scegliere più intelligentemente, in dipendenza del problema, il linguaggio di programmazione.

Introduzione al corso

Obiettivi specifici

Obiettivi generali

Scheduling

Altre informazioni

Linguaggi (di  
programmazione)

Macchine astratte

Paradigmi  
computazionali



# Obiettivi generali

- Migliorare l'abilità nel risolvere i problemi usando meglio i linguaggi di programmazione.
- Imparare a scegliere più intelligentemente, in dipendenza del problema, il linguaggio di programmazione.
- Aumentare la capacità di imparare linguaggi di programmazione (che sono TANTI!)

[Introduzione al corso](#)

[Obiettivi specifici](#)

**Obiettivi generali**

[Scheduling](#)

[Altre informazioni](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)



# Scheduling

Introduzione al corso

Obiettivi specifici

Obiettivi generali

**Scheduling**

Altre informazioni

Linguaggi (di  
programmazione)

Macchine astratte

Paradigmi  
computazionali

## Parte prima

1. Storia e concetti di base
2. Primi cenni ai paradigmi dei linguaggi di programmazione.
3. Il modello imperativo.



# Scheduling

Introduzione al corso

Obiettivi specifici

Obiettivi generali

**Scheduling**

Altre informazioni

Linguaggi (di  
programmazione)

Macchine astratte

Paradigmi  
computazionali

## ■ Parte prima

1. Storia e concetti di base
2. Primi cenni ai paradigmi dei linguaggi di programmazione.
3. Il modello imperativo.

## ■ Parte seconda (modello object-oriented, Java)

1. Studio dei costrutti più avanzati: qualificatori di classi, metodi e attributi; classi astratte; interfacce; template; classi interne; gestione degli errori: eccezioni.



# Scheduling

Introduzione al corso

Obiettivi specifici

Obiettivi generali

**Scheduling**

Altre informazioni

Linguaggi (di  
programmazione)

Macchine astratte

Paradigmi  
computazionali

## ■ Parte prima

1. Storia e concetti di base
2. Primi cenni ai paradigmi dei linguaggi di programmazione.
3. Il modello imperativo.

## ■ Parte seconda (modello object-oriented, Java)

1. Studio dei costrutti più avanzati: qualificatori di classi, metodi e attributi; classi astratte; interfacce; template; classi interne; gestione degli errori: eccezioni.

## ■ Parte terza (Linguaggi funzionali, ML)

1. ML e costrutti funzionali: funzioni di ordine superiore; polimorfismo; tipi di dato astratti e interfacce; template; gestione degli errori: eccezioni; type inference.



# Scheduling

Introduzione al corso

Obiettivi specifici

Obiettivi generali

**Scheduling**

Altre informazioni

Linguaggi (di programmazione)

Macchine astratte

Paradigmi computazionali

## ■ Parte prima

1. Storia e concetti di base
2. Primi cenni ai paradigmi dei linguaggi di programmazione.
3. Il modello imperativo.

## ■ Parte seconda (modello object-oriented, Java)

1. Studio dei costrutti più avanzati: qualificatori di classi, metodi e attributi; classi astratte; interfacce; template; classi interne; gestione degli errori: eccezioni.

## ■ Parte terza (Linguaggi funzionali, ML)

1. ML e costrutti funzionali: funzioni di ordine superiore; polimorfismo; tipi di dato astratti e interfacce; template; gestione degli errori: eccezioni; type inference.

## ■ Parte quarta (Linguaggi logici, Prolog)

1. Costrutti fondamentali: termini, predicati e regole.
2. Tecniche di programmazione avanzate: invertibilità e nondeterminismo.



## Altre informazioni

### ■

### *Libri di testo:*

- ◆ Gabbrielli - Martini. *Linguaggi di programmazione: Principi e paradigmi*
- ◆ Slide e dispense fornite dal docente

Introduzione al corso

Obiettivi specifici

Obiettivi generali

Scheduling

**Altre informazioni**

Linguaggi (di  
programmazione)

Macchine astratte

Paradigmi  
computazionali



## Altre informazioni

Introduzione al corso

Obiettivi specifici

Obiettivi generali

Scheduling

**Altre informazioni**

Linguaggi (di  
programmazione)

Macchine astratte

Paradigmi  
computazionali

### ■ *Libri di testo:*

- ◆ Gabbrielli - Martini. *Linguaggi di programmazione: Principi e paradigmi*
- ◆ Slide e dispense fornite dal docente

■ *Materiali:* Sul gruppo Teams e su <http://wpage.unina.it/mfaella>

■ *Ricevimento studenti:* in presenza o in remoto nel gruppo Teams  
“Ricevimento Faella” (orario sulla homepage)



Introduzione al corso

Linguaggi (di  
programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti

introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali

# Linguaggi (di programmazione)



# Sono ~ 40?

Introduzione al corso

Linguaggi (di programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

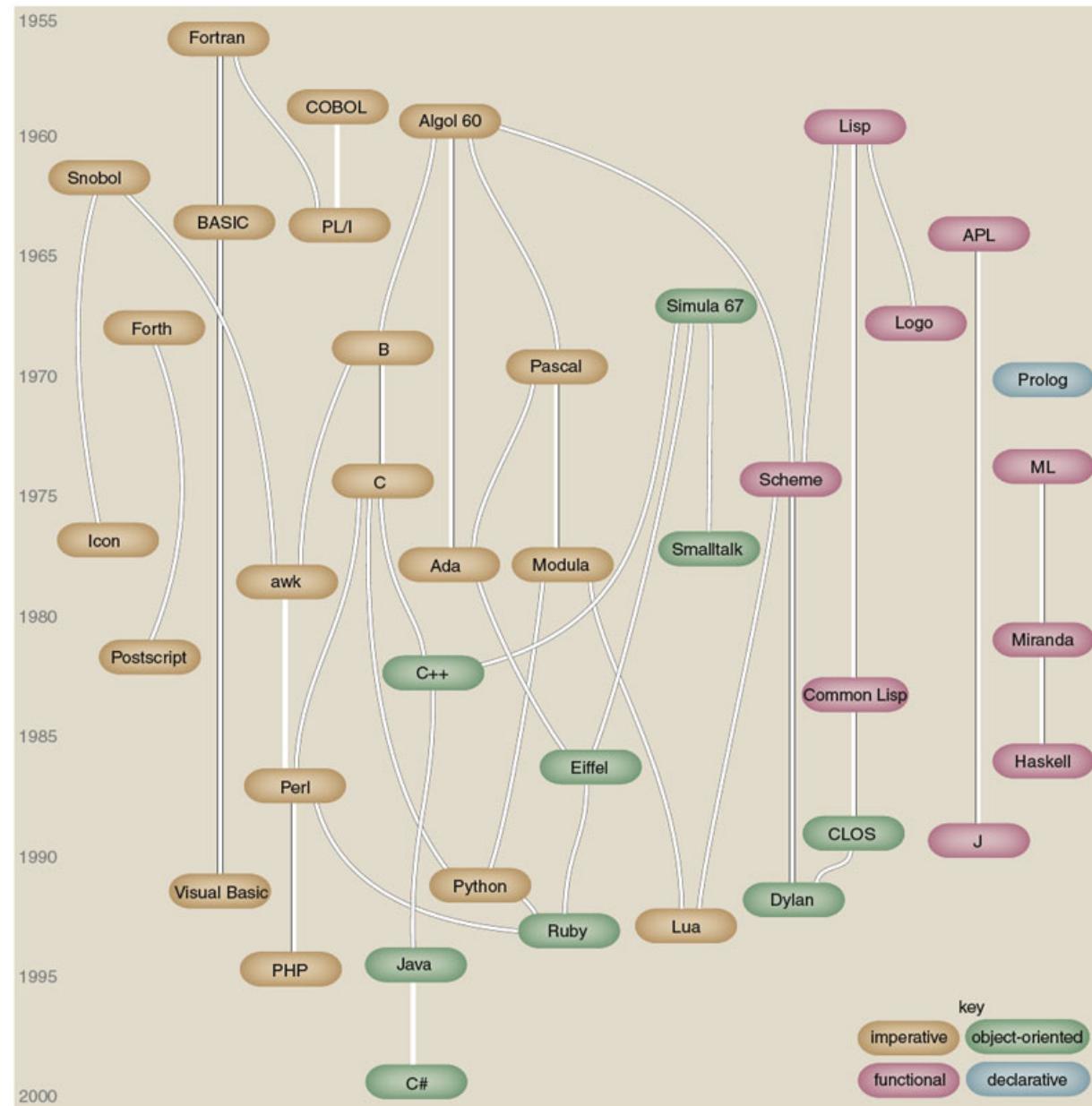
. . . concetti  
introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali





# Sono ~ 80?

Introduzione al corso

Linguaggi (di programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi computazionali

## Mother Tongues

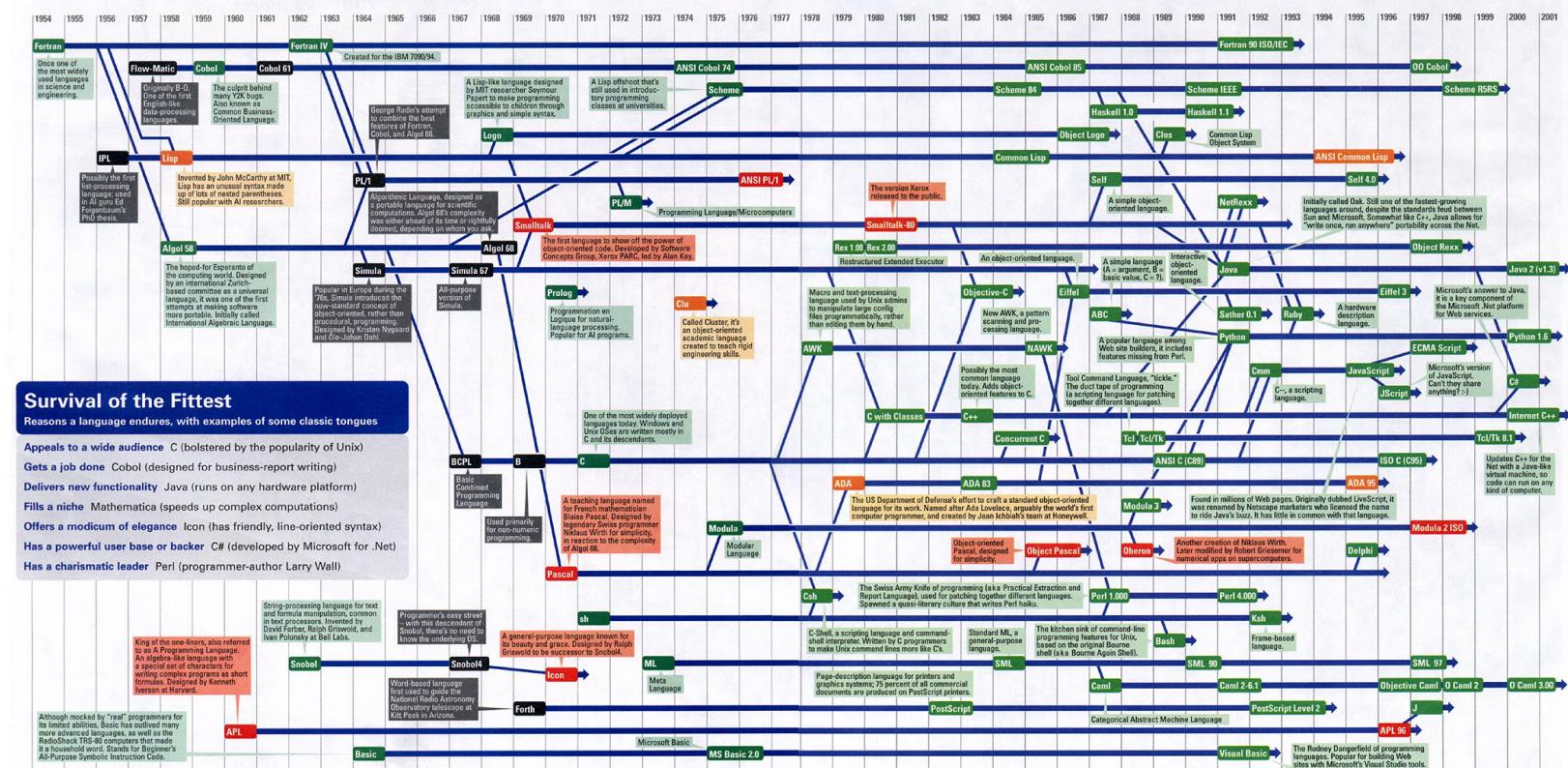
Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java, and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers—electronic lexicographers, if you will—aim to save, or at least document, the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at [www.informatik.uni-freiburg.de/Java/misc/lang\\_list.html](http://www.informatik.uni-freiburg.de/Java/misc/lang_list.html) — Michael Menduno

Key	
1954	Year Introduced
Green	Active: thousands of users
Orange	Protected: taught at universities; compilers available
Red	Endangered: usage dropping off
Black	Extinct: no known active users or up-to-date compilers
Blue arrow	Lineage continues





# Quanti sono?

Introduzione al corso

Linguaggi (di programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

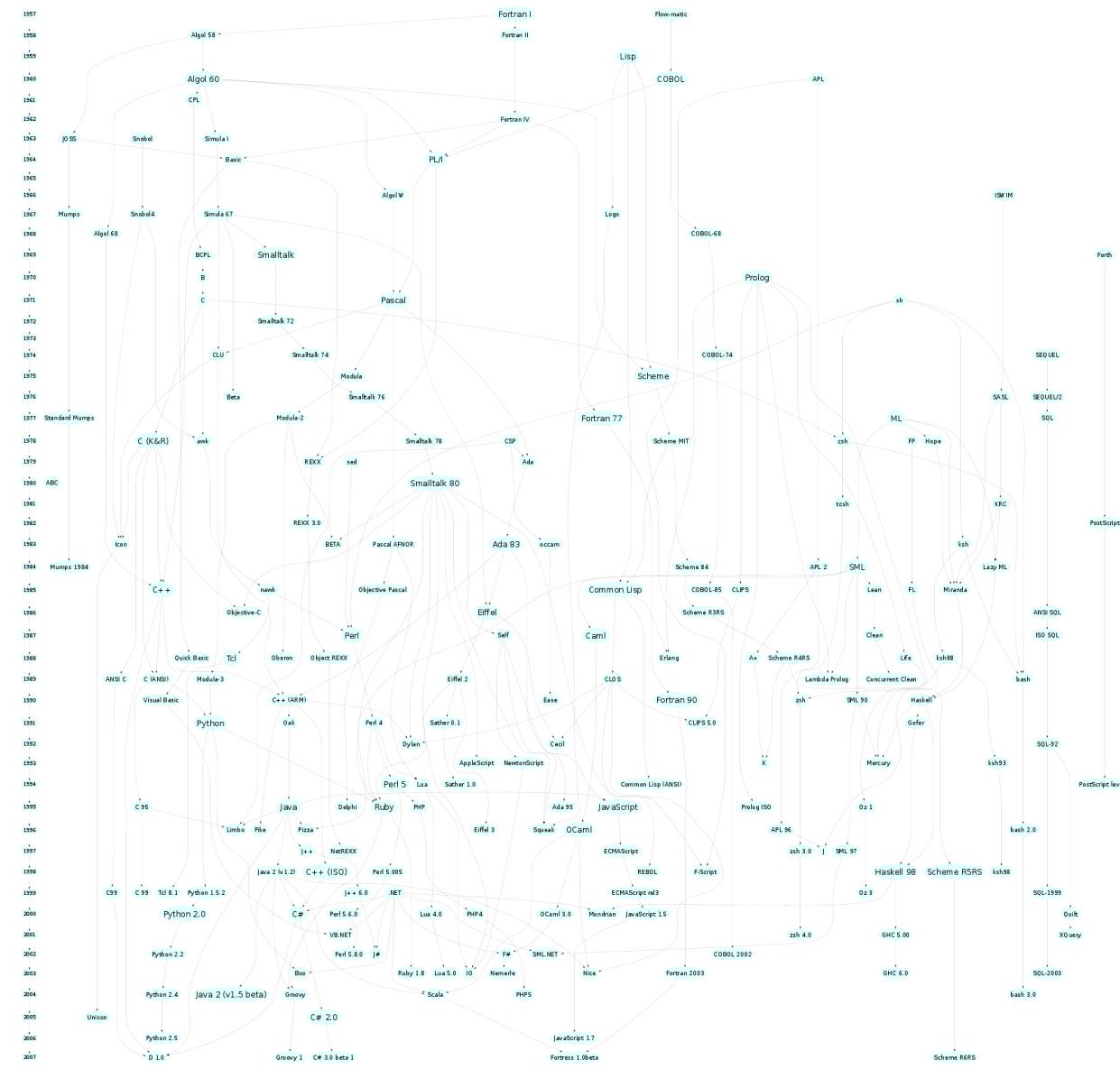
. . . concetti  
introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali





# Quanti sono?

- C'è chi dice addirittura qualche migliaio
  - ◆ Come orientarsi?
  - ◆ Come usarli *bene*?
  - ◆ Come apprendere in fretta quelli nuovi?
- Occorre comprensione astratta delle caratteristiche dei linguaggi per coglierne somiglianze/differenze
- e per comprendere lo scopo di ciascun costrutto (ovvero i principi del *language design*)

Introduzione al corso

Linguaggi (di  
programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti  
introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali



## Breve storia

1954

FORTRAN (FORmula TRANslation)

Introduzione al corso

Linguaggi (di  
programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti  
introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali



## Breve storia

Introduzione al corso

Linguaggi (di  
programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

**Breve storia**

Storia dei . . .

. . . concetti  
introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali

1954	FORTRAN (FORmula TRANslation)
1960	COBOL (Common Business Oriented Language) ALGOL 60 (Algorithmic Oriented Language) PL/1 (Programming Language 1) Simula 67 ALGOL 68 PASCAL LISP (LISt Processing) APL BASIC



## Breve storia

Introduzione al corso

Linguaggi (di  
programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

**Breve storia**

Storia dei . . .

. . . concetti  
introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali

1954	FORTRAN (FORmula TRANslation)
1960	COBOL (Common Business Oriented Language) ALGOL 60 (Algorithmic Oriented Language) PL/1 (Programming Language 1) Simula 67 ALGOL 68 PASCAL LISP (LISt Processing) APL BASIC
1970/80	PROLOG SMALLTALK C MODULA/2 ADA



## Storia dei concetti introdotti dai progenitori dell'oggi

**Fortran:** FORmula TRANslator. Nato per calcolo scientifico. Introduce: variabili, statement di assegnazione, concetto di tipo, subroutine, iterazione e statement condizionali, go-to, formati di input e output.

Gestione solo statica della memoria, no ricorsione, no strutture dinamiche, no tipi definiti da utente.



## Storia dei concetti introdotti dai progenitori dell'oggi

**Fortran:** FORmula TRANslator. Nato per calcolo scientifico. Introduce: variabili, statement di assegnazione, concetto di tipo, subroutine, iterazione e statement condizionali, go-to, formati di input e output.

Gestione solo statica della memoria, no ricorsione, no strutture dinamiche, no tipi definiti da utente.

**Cobol:** indipendenza dalla macchina e statement “English like”. Orientato ai database. Introduce il record. Tuttora usato in ambito bancario.



## Storia dei concetti introdotti dai progenitori dell'oggi

**Fortran:** FORmula TRANslator. Nato per calcolo scientifico. Introduce: variabili, statement di assegnazione, concetto di tipo, subroutine, iterazione e statement condizionali, go-to, formati di input e output.

Gestione solo statica della memoria, no ricorsione, no strutture dinamiche, no tipi definiti da utente.

**Cobol:** indipendenza dalla macchina e statement “English like”. Orientato ai database. Introduce il record. Tuttora usato in ambito bancario.

**Algol60:** indipendenza dalla macchina e definizione mediante grammatica (*Backus-Naur form*), strutture a blocco, supporto generale dell’iterazione e ricorsione.



# Storia dei concetti introdotti dai progenitori dell'oggi

**Fortran:** FORmula TRANslator. Nato per calcolo scientifico. Introduce: variabili, statement di assegnazione, concetto di tipo, subroutine, iterazione e statement condizionali, go-to, formati di input e output.

Gestione solo statica della memoria, no ricorsione, no strutture dinamiche, no tipi definiti da utente.

**Cobol:** indipendenza dalla macchina e statement “English like”. Orientato ai database. Introduce il record. Tuttora usato in ambito bancario.

**Algol60:** indipendenza dalla macchina e definizione mediante grammatica (*Backus-Naur form*), strutture a blocco, supporto generale dell’iterazione e ricorsione.

**Lisp:** primo vero linguaggio di manipolazione simbolica, paradigma **funzionale**, non c’è lo statement di assegnazione, e quindi concettualmente non c’è “il valore” ovvero l’idea di cambiare lo stato della memoria. Non c’è differenza concettuale fra funzione e dato: dipende dall’uso. La prima versione era essenzialmente non tipata.



# Storia dei concetti introdotti dai progenitori dell'oggi

**Prolog:** primo (e principale) linguaggio di programmazione logica (paradigma logico). Tra le caratteristiche innovative: invertibilità, programmazione in stile nondeterministico (*generate and test*). Essenzialmente non tipato; estensioni (tipi e altro) mediante metaprogrammazione.



# Storia dei concetti introdotti dai progenitori dell'oggi

**Prolog:** primo (e principale) linguaggio di programmazione logica (paradigma logico). Tra le caratteristiche innovative: invertibilità, programmazione in stile nondeterministico (*generate and test*). Essenzialmente non tipato; estensioni (tipi e altro) mediante metaprogrammazione.

**Simula 67:** classe come encapsulamento di dati e procedure, istanze delle classi (oggetti): anticipatorio del concetto di tipo di dato

astratto implementato in Ada e Modula2, e del concetto moderno di classe di Smalltalk e C++.



# Storia dei concetti introdotti dai progenitori dell'oggi

**Prolog:** primo (e principale) linguaggio di programmazione logica (paradigma logico). Tra le caratteristiche innovative: invertibilità, programmazione in stile nondeterministico (*generate and test*). Essenzialmente non tipato; estensioni (tipi e altro) mediante metaprogrammazione.

**Simula 67:** classe come encapsulamento di dati e procedure, istanze delle classi (oggetti): anticipatorio del concetto di tipo di dato

astratto implementato in Ada e Modula2, e del concetto moderno di classe di Smalltalk e C++.

**PL/1:** abilità ad eseguire procedure specificate quando si verifica una condizione eccezionale; “multitasking”, cioè specificazione di tasks che possono essere eseguiti in concorrenza.



# Storia dei concetti introdotti dai progenitori dell'oggi

**Prolog:** primo (e principale) linguaggio di programmazione logica (paradigma logico). Tra le caratteristiche innovative: invertibilità, programmazione in stile nondeterministico (*generate and test*). Essenzialmente non tipato; estensioni (tipi e altro) mediante metaprogrammazione.

**Simula 67:** classe come encapsulamento di dati e procedure, istanze delle classi (oggetti): anticipatorio del concetto di tipo di dato

astratto implementato in Ada e Modula2, e del concetto moderno di classe di Smalltalk e C++.

**PL/1:** abilità ad eseguire procedure specificate quando si verifica una condizione eccezionale; “multitasking”, cioè specificazione di tasks che possono essere eseguiti in concorrenza.

**Pascal:** programmazione strutturata, tipi di dato definiti da utente, ricchezza di strutture dati. Ma ancora niente encapsulation; si dovrà aspettare Modula.



# Terminologia

- **Linguaggio di programmazione:** è un linguaggio che è usato per esprimere (mediante un programma) un algoritmo con il quale un processore può risolvere un problema.

Introduzione al corso

Linguaggi (di  
programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti

introdotti

**Terminologia**

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali



# Terminologia

- **Linguaggio di programmazione:** è un linguaggio che è usato per esprimere (mediante un programma) un algoritmo con il quale un processore può risolvere un problema.
- **Programma:** è la codifica di un algoritmo in un linguaggio di programmazione.
- **Processore:** è la macchina (reale o virtuale) che eseguirà l'algoritmo descritto dal programma. Non si deve intendere come un singolo oggetto, ma come una *architettura di elaborazione*.

Introduzione al corso

Linguaggi (di  
programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti  
introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali



# Terminologia

- **Linguaggio di programmazione:** è un linguaggio che è usato per esprimere (mediante un programma) un algoritmo con il quale un processore può risolvere un problema.
- **Programma:** è la codifica di un algoritmo in un linguaggio di programmazione.
- **Processore:** è la macchina (reale o virtuale) che eseguirà l'algoritmo descritto dal programma. Non si deve intendere come un singolo oggetto, ma come una *architettura di elaborazione*.

Introduzione al corso

Linguaggi (di  
programmazione)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti  
introdotti

Terminologia

Linguaggi completi

Macchine astratte

Paradigmi  
computazionali



# Linguaggi completi

- È uso comune intendere come linguaggi di programmazione solo quelli *computazionalmente completi*, cioè solo quelli che possono esprimere *qualunque funzione calcolabile*
  - ◆ detti anche *general purpose*
  - ◆ tecnicamente: devono poter simulare *qualunque macchina di Turing*.

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti

introdotti

Terminologia

**Linguaggi completi**

[Macchine astratte](#)

[Paradigmi computazionali](#)



# Linguaggi completi

- È uso comune intendere come linguaggi di programmazione solo quelli *computazionalmente completi*, cioè solo quelli che possono esprimere *qualunque funzione calcolabile*
  - ◆ detti anche *general purpose*
  - ◆ tecnicamente: devono poter simulare qualunque *macchina di Turing*.
- Sono completi solo quelli che riescono ad esprimere anche programmi di cui non è decidibile la terminazione

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti

introdotti

Terminologia

[Linguaggi completi](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)



# Linguaggi completi

- È uso comune intendere come linguaggi di programmazione solo quelli *computazionalmente completi*, cioè solo quelli che possono esprimere *qualunque funzione calcolabile*
  - ◆ detti anche *general purpose*
  - ◆ tecnicamente: devono poter simulare qualunque *macchina di Turing*.
- Sono completi solo quelli che riescono ad esprimere anche programmi di cui non è decidibile la terminazione
- SQL non è un linguaggio completo (perché la terminazione dei programmi è sempre decidibile).

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti

introdotti

Terminologia

[Linguaggi completi](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)



# Linguaggi completi

- È uso comune intendere come linguaggi di programmazione solo quelli *computazionalmente completi*, cioè solo quelli che possono esprimere *qualunque funzione calcolabile*
  - ◆ detti anche *general purpose*
  - ◆ tecnicamente: devono poter simulare qualunque *macchina di Turing*.
- Sono completi solo quelli che riescono ad esprimere anche programmi di cui non è decidibile la terminazione
- SQL non è un linguaggio completo (perché la terminazione dei programmi è sempre decidibile). È spesso immerso in linguaggi completi.

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti

introdotti

Terminologia

[Linguaggi completi](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)



# Linguaggi completi

- È uso comune intendere come linguaggi di programmazione solo quelli *computazionalmente completi*, cioè solo quelli che possono esprimere *qualunque funzione calcolabile*
  - ◆ detti anche *general purpose*
  - ◆ tecnicamente: devono poter simulare qualunque *macchina di Turing*.
- Sono completi solo quelli che riescono ad esprimere anche programmi di cui non è decidibile la terminazione
- SQL non è un linguaggio completo (perché la terminazione dei programmi è sempre decidibile). È spesso immerso in linguaggi completi.
- HTML non è un linguaggio completo (idem).
- Per mostrare la completezza di un linguaggio: usarlo per simulare arbitrarie macchine di Turing

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

Sono ~ 40?

Sono ~ 80?

Quanti sono?

Quanti sono?

Breve storia

Storia dei . . .

. . . concetti

introdotti

Terminologia

[Linguaggi completi](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)



[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

**Macchine astratte**

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

# Macchine astratte



## Definizione

Dato un linguaggio di programmazione  $L$ , una macchina astratta per  $L$  (in simboli,  $M_L$ ) è un qualsiasi insieme di strutture dati e algoritmi che permettano di memorizzare ed eseguire programmi scritti in  $L$ .

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

**Definizione**

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi computazionali



# Definizione

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

**Definizione**

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio → Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un LP

Proprietà dei ...

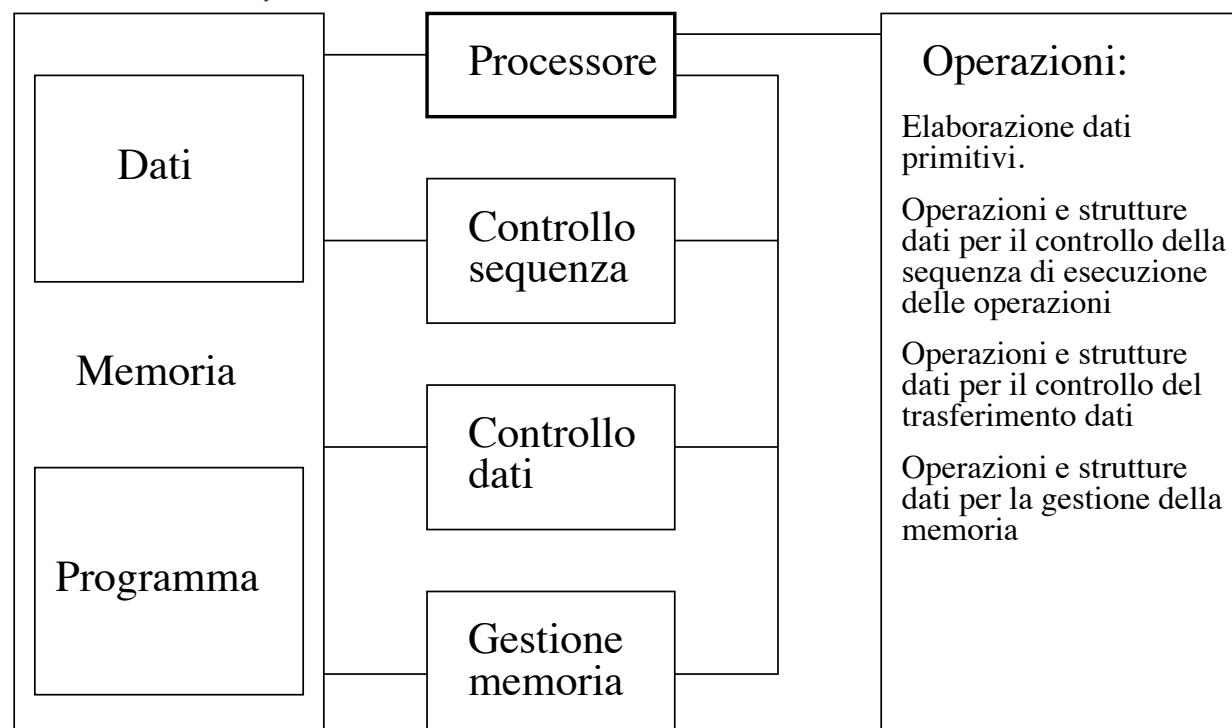
Criteri di scelta ...

Paradigmi

computazionali

Dato un linguaggio di programmazione  $L$ , una macchina astratta per  $L$  (in simboli,  $M_L$ ) è un qualsiasi insieme di strutture dati e algoritmi che permettano di memorizzare ed eseguire programmi scritti in  $L$ .

La struttura di una macchina astratta è (essenzialmente memoria e processore):





# Esempio

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

**Esempio**

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi

computazionali

## Macchina E–Business (commercio online)

### Macchina Web Service

#### Macchina Web

#### Macchina linguaggio di programmazione

#### Macchina intermedia (java bytecode)

#### Macchina Sistema Operativo

#### Macchina firmware

#### Macchina hardware



# Processore della macchina astratta

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

**Processore M.A.**

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →  
Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

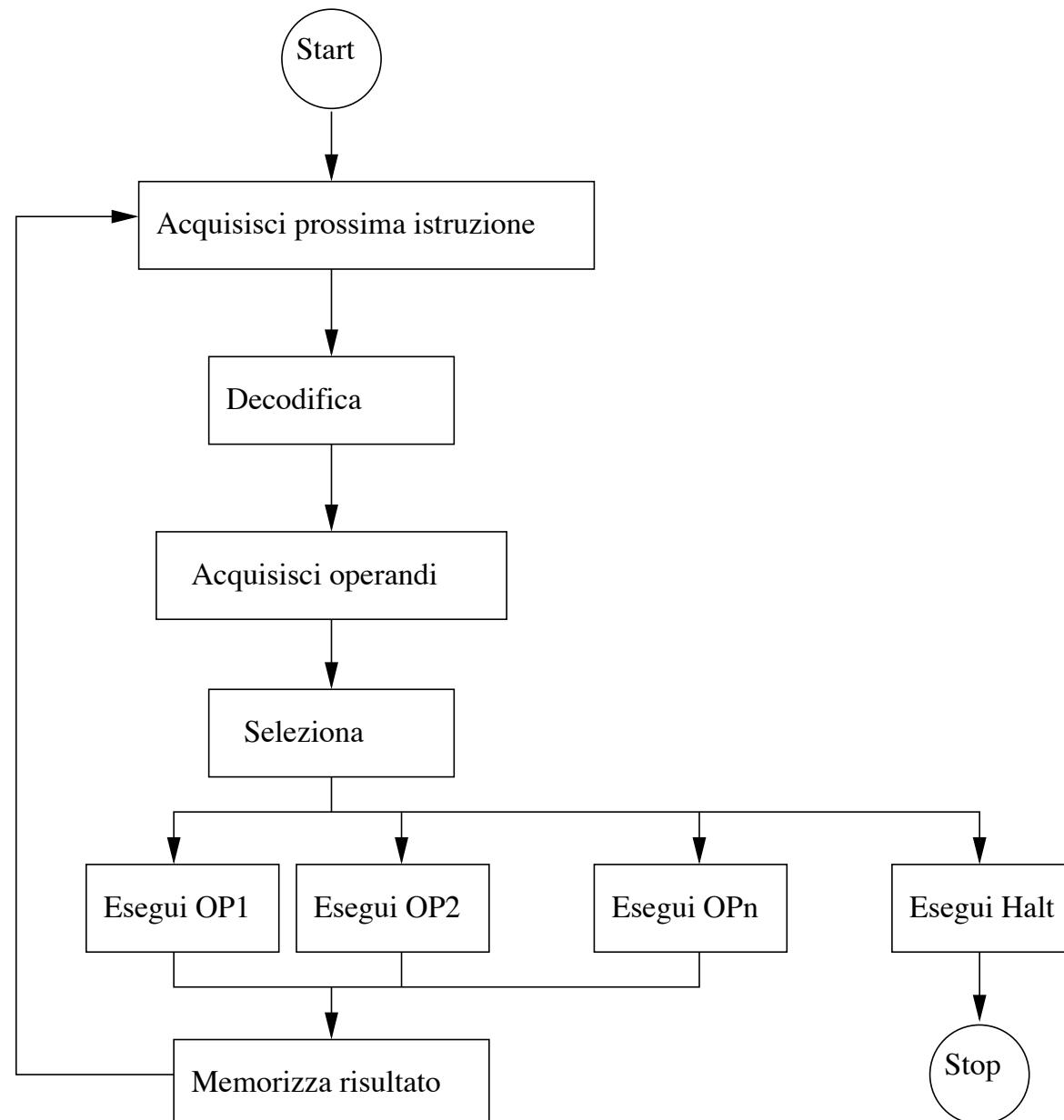
Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali





# Esempi di macchine astratte

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

**Esempi M.A.**

Realizzazione M.A.

Traduttori

Linguaggio →  
Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

Linguaggio	Macchina astratta
C	gcc + OS + HW
Java	javac + JVM + OS + HW
C#	csc + CLR + OS + HW
Python	python + OS + HW



# Tecnologie di realizzazione di macchina astratta

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

Definizione

Esempio

Processore M.A.

Esempi M.A.

**Realizzazione M.A.**

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

**Hardware:** Si era pensato per Lisp e Prolog



# Tecnologie di realizzazione di macchina astratta

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

Definizione

Esempio

Processore M.A.

Esempi M.A.

**Realizzazione M.A.**

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi

computazionali

**Hardware:** Si era pensato per Lisp e Prolog

**Firmware:** Soluzione più flessibile e semplice / economicità di progetto



# Tecnologie di realizzazione di macchina astratta

[Introduzione al corso](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

Definizione

Esempio

Processore M.A.

Esempi M.A.

**Realizzazione M.A.**

Traduttori

Linguaggio →  
Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

**Hardware:** Si era pensato per Lisp e Prolog

**Firmware:** Soluzione più flessibile e semplice / economicità di progetto

**Software:** Ad es. Java Virtual Machine, o Warren Abstract Machine (Prolog)



# Traduttori Linguaggio → Macchina astratta

- **Interpreti:** traducono ed eseguono un costrutto alla volta.  
PRO: debug - fase di sviluppo: interazione più snella.

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali



# Traduttori Linguaggio → Macchina astratta

- **Interpreti:** traducono ed eseguono un costrutto alla volta.  
PRO: debug - fase di sviluppo: interazione più snella.
- **Compilatori:** prima traducono l'intero programma; poi la traduzione può essere eseguita (anche più volte).  
PRO: velocità di esecuzione finale - fase di rilascio.  
PRO: più controlli e in anticipo

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori  
Linguaggio →  
Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali



# Interpretazione pura

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →  
Macchina astratta

**Interpretazione pura**

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

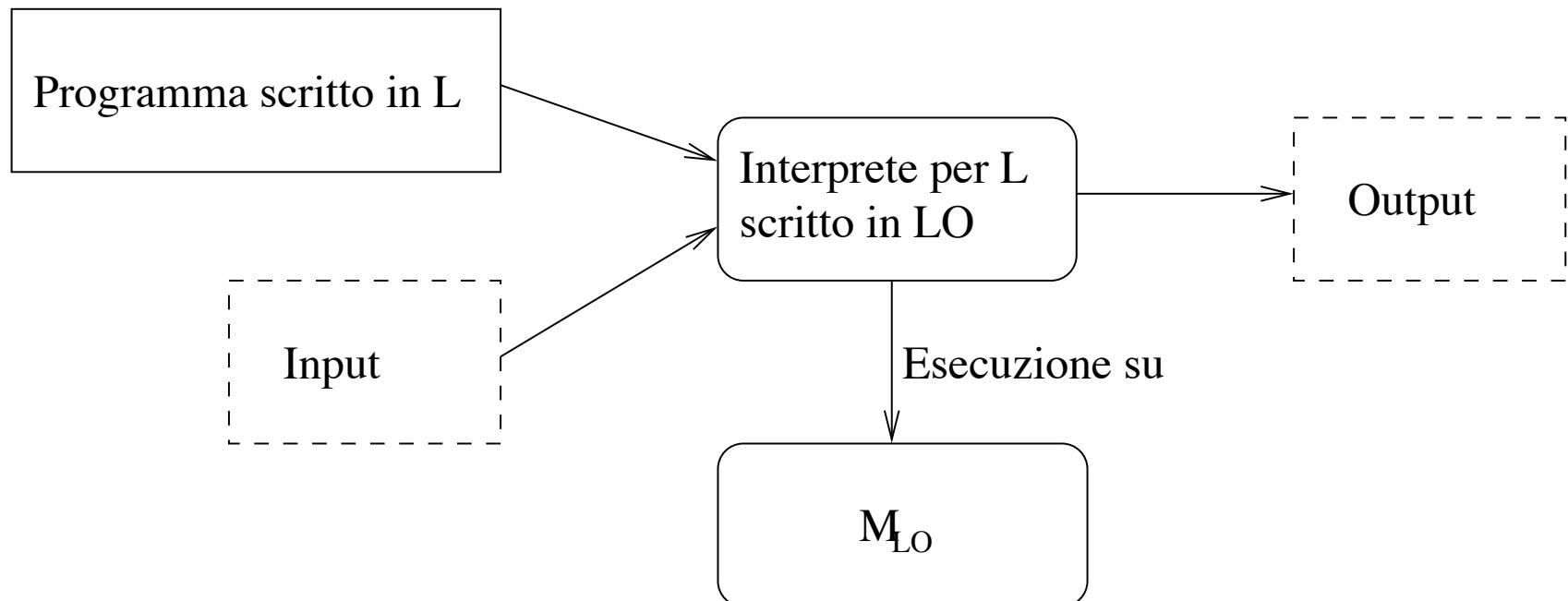
Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali



E' il caso dei linguaggi di scripting: bash, awk, Python, etc.



# Compilazione pura (caso semplice)

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio → Macchina astratta

Interpretazione pura

**Compilazione 1**

Compilazione 2

Compilazione 3

Compilazione ed ...

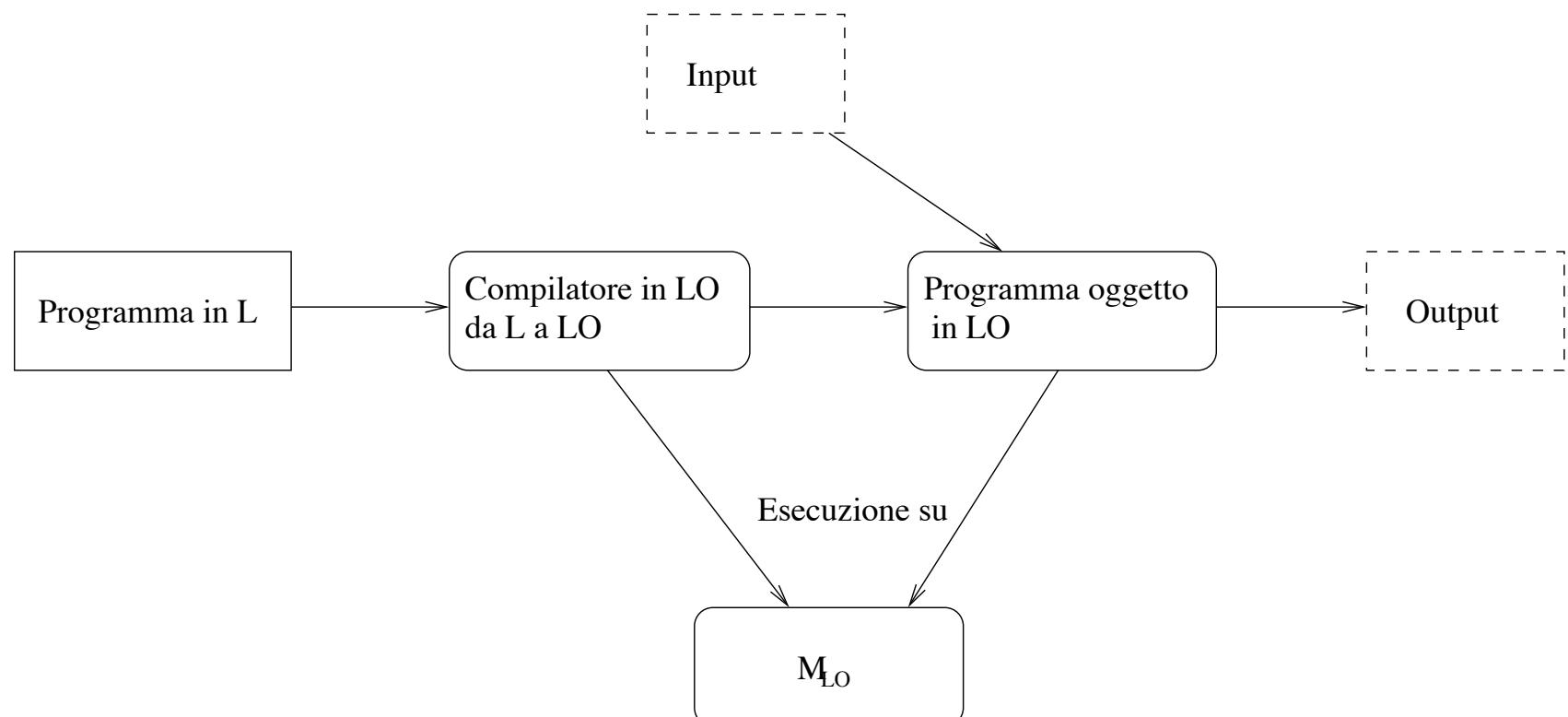
Compilatore

Componenti di un LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi computazionali





# Compilazione pura (caso più generale)

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio → Macchina astratta

Interpretazione pura

Compilazione 1

**Compilazione 2**

Compilazione 3

Compilazione ed ...

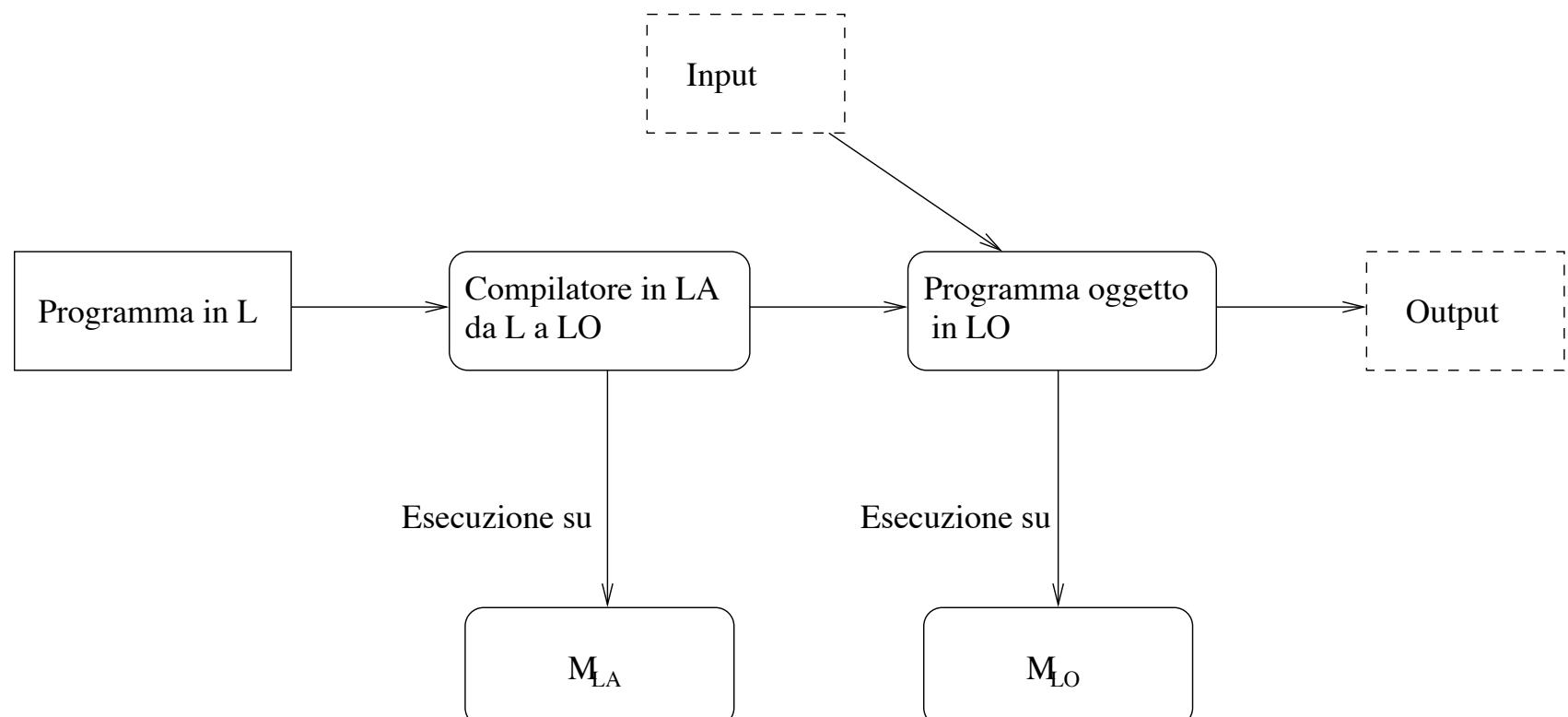
Compilatore

Componenti di un LP

Proprietà dei ...

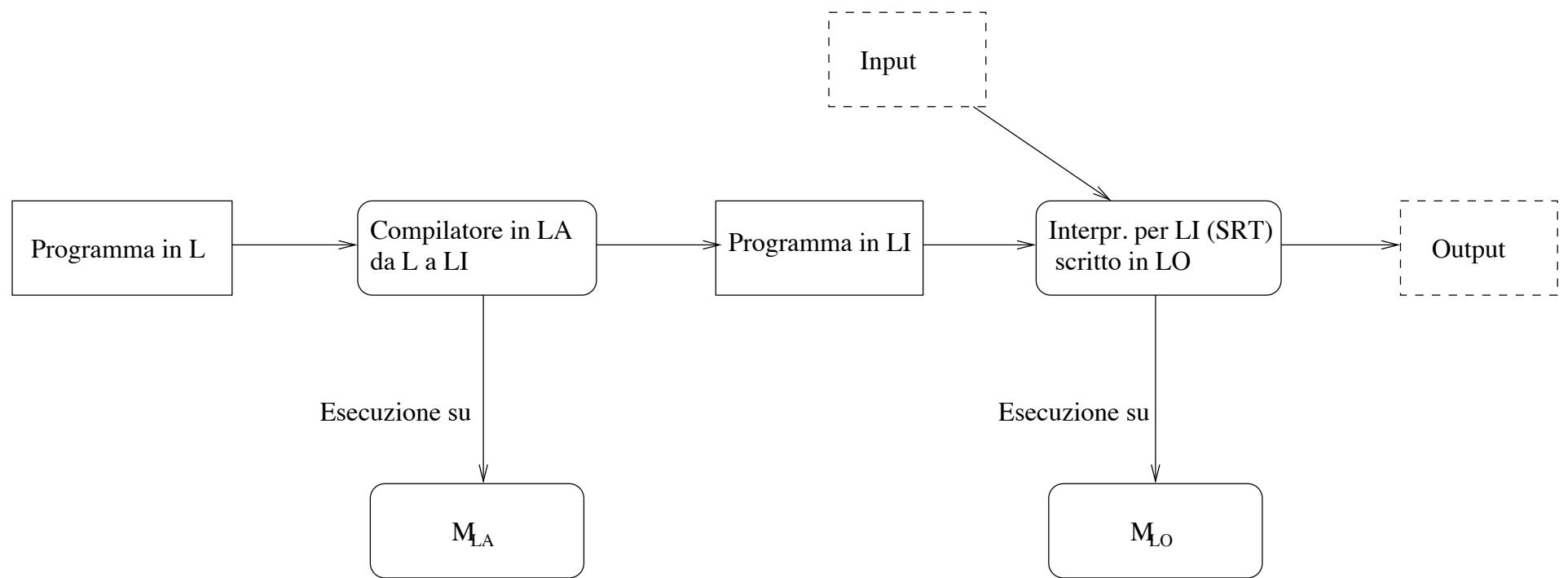
Criteri di scelta ...

Paradigmi computazionali





# Compilazione per macchina intermedia



Esercizio: Se L è il linguaggio Java, chi sono LA, LI, e LO?



# Compilazione ed esecuzione

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio → Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

**Compilazione ed ...**

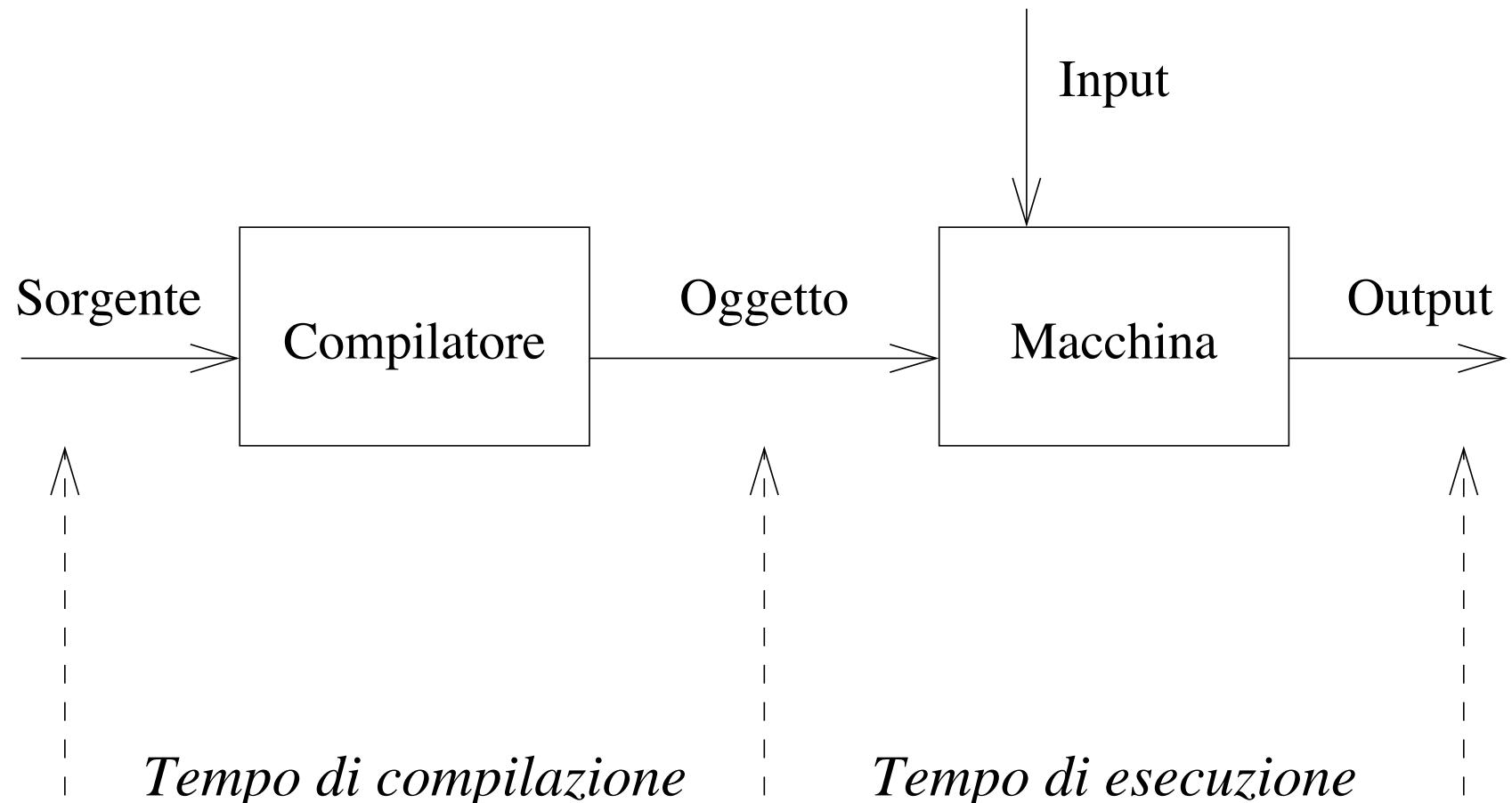
Compilatore

Componenti di un LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi computazionali





# Compilatore

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

**Compilatore**

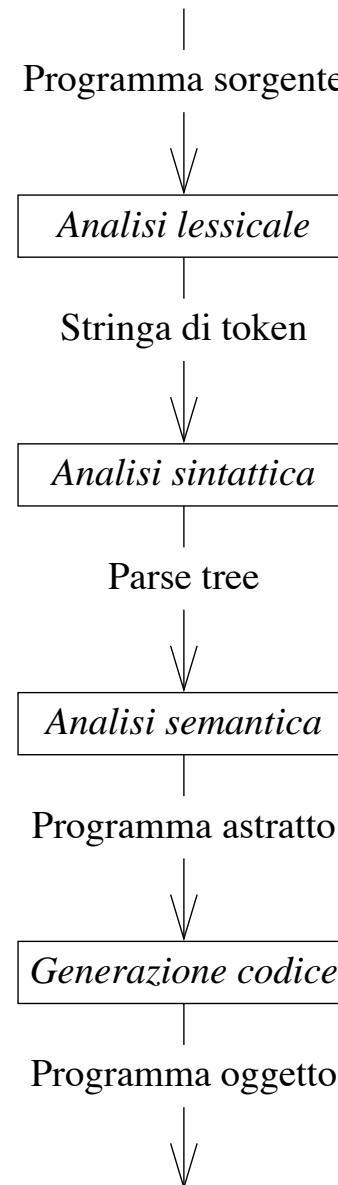
Componenti di un

LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali





# Componenti di un LP

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

## ■ Grammatica

- ◆ *Come si scrive un programma ben formato (cioè, eseguibile)?*
- ◆ *Anche detta sintassi*



# Componenti di un LP

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

## ■ Grammatica

- ◆ *Come si scrive un programma ben formato (cioè, eseguibile)?*

- ◆ Anche detta *sintassi*

## ■ Semantica

- ◆ *Qual è il significato di ciascuna istruzione/costrutto?*



# Componenti di un LP

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

## ■ Grammatica

- ◆ *Come si scrive un programma ben formato (cioè, eseguibile)?*

- ◆ Anche detta *sintassi*

## ■ Semantica

- ◆ *Qual è il significato di ciascuna istruzione/costrutto?*

## ■ Pragmatica

- ◆ *Come si usa efficacemente il linguaggio? (Programmazione & Ingegneria del Software)*



# Componenti di un LP

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

## ■ Grammatica

- ◆ *Come si scrive un programma ben formato (cioè, eseguibile)?*

- ◆ Anche detta *sintassi*

## ■ Semantica

- ◆ *Qual è il significato di ciascuna istruzione/costrutto?*

## ■ Pragmatica

- ◆ *Come si usa efficacemente il linguaggio? (Programmazione & Ingegneria del Software)*

## ■ Implementazione

- ◆ *Come si realizza una macchina astratta per questo linguaggio? (Costruzione di compilatori)*

In questo corso, ci concentreremo su grammatica e semantica, con accenni all'implementazione



# Proprietà dei linguaggi

- Semplicità – (concisione) VS (leggibilità)
  - ◆ Sintattica: unica rappresentabilità di ogni concetto.
  - ◆ Semantica: minimo numero di concetti e strutture.

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali



# Proprietà dei linguaggi

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi computazionali

- Semplicità – (concisione) VS (leggibilità)
  - ◆ Sintattica: unica rappresentabilità di ogni concetto.
  - ◆ Semantica: minimo numero di concetti e strutture.
- Astrazione – (incapsulare e dividere)
  - ◆ Dati: nascondere i dettagli (incapsulamento, information hiding)
  - ◆ Funzioni e procedure: dividere il programma
  - ◆ Moduli, pacchetti, etc.: favorire la modularità



# Proprietà dei linguaggi

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi  
computazionali

- Semplicità – (concisione) VS (leggibilità)
  - ◆ Sintattica: unica rappresentabilità di ogni concetto.
  - ◆ Semantica: minimo numero di concetti e strutture.
- Astrazione – (incapsulare e dividere)
  - ◆ Dati: nascondere i dettagli (incapsulamento, information hiding)
  - ◆ Funzioni e procedure: dividere il programma
  - ◆ Moduli, pacchetti, etc.: favorire la modularità
- Espressività – (facilità di rappresentazione di oggetti) VS (semplicità)



# Proprietà dei linguaggi

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi computazionali

- Semplicità – (concisione) VS (leggibilità)
  - ◆ Sintattica: unica rappresentabilità di ogni concetto.
  - ◆ Semantica: minimo numero di concetti e strutture.
- Astrazione – (incapsulare e dividere)
  - ◆ Dati: nascondere i dettagli (incapsulamento, information hiding)
  - ◆ Funzioni e procedure: dividere il programma
  - ◆ Moduli, pacchetti, etc.: favorire la modularità
- Espressività – (facilità di rappresentazione di oggetti) VS (semplicità)
- Ortogonalità – (poche eccezioni alle regole del linguaggio)



# Proprietà dei linguaggi

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un LP

Proprietà dei ...

Criteri di scelta ...

Paradigmi computazionali

- Semplicità – (concisione) VS (leggibilità)
  - ◆ Sintattica: unica rappresentabilità di ogni concetto.
  - ◆ Semantica: minimo numero di concetti e strutture.
- Astrazione – (incapsulare e dividere)
  - ◆ Dati: nascondere i dettagli (incapsulamento, information hiding)
  - ◆ Funzioni e procedure: dividere il programma
  - ◆ Moduli, pacchetti, etc.: favorire la modularità
- Espressività – (facilità di rappresentazione di oggetti) VS (semplicità)
- Ortogonalità – (poche eccezioni alle regole del linguaggio)
- Portabilità



# Criteri di scelta del linguaggio

## ■ Disponibilità di interpreti/compilatori

Introduzione al corso

Linguaggi (di programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

**Criteri di scelta ...**

Paradigmi  
computazionali



# Criteri di scelta del linguaggio

- Disponibilità di interpreti/compilatori
- Esistenza di standard di portabilità

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

**Criteri di scelta ...**

Paradigmi  
computazionali



# Criteri di scelta del linguaggio

- Disponibilità di interpreti/compilatori
- Esistenza di standard di portabilità
- Interoperabilità con altri sistemi e linguaggi

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

**Criteri di scelta ...**

Paradigmi  
computazionali



# Criteri di scelta del linguaggio

- Disponibilità di interpreti/compilatori
- Esistenza di standard di portabilità
- Interoperabilità con altri sistemi e linguaggi
- Esistenza di librerie specifiche

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

**Criteri di scelta ...**

Paradigmi  
computazionali



# Criteri di scelta del linguaggio

- Disponibilità di interpreti/compilatori
- Esistenza di standard di portabilità
- Interoperabilità con altri sistemi e linguaggi
- Esistenza di librerie specifiche
- Maggiore conoscenza da parte del programmatore

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →  
Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

**Criteri di scelta ...**

Paradigmi  
computazionali



# Criteri di scelta del linguaggio

■ Disponibilità di interpreti/compilatori

■ Esistenza di standard di portabilità

■ Interoperabilità con altri sistemi e linguaggi

■ Esistenza di librerie specifiche

■ Maggiore conoscenza da parte del programmatore

■ Comodità dell'ambiente di programmazione (IDE)

■ Sintassi aderente al problema

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un  
LP

Proprietà dei ...

**Criteri di scelta ...**

Paradigmi  
computazionali



# Criteri di scelta del linguaggio

■ Disponibilità di interpreti/compilatori

■ Esistenza di standard di portabilità

■ Interoperabilità con altri sistemi e linguaggi

■ Esistenza di librerie specifiche

■ Maggiore conoscenza da parte del programmatore

■ Comodità dell'ambiente di programmazione (IDE)

■ Sintassi aderente al problema

Introduzione al corso

Linguaggi (di  
programmazione)

Macchine astratte

Definizione

Esempio

Processore M.A.

Esempi M.A.

Realizzazione M.A.

Traduttori

Linguaggio →

Macchina astratta

Interpretazione pura

Compilazione 1

Compilazione 2

Compilazione 3

Compilazione ed ...

Compilatore

Componenti di un

LP

Proprietà dei ...

**Criteri di scelta ...**

Paradigmi  
computazionali



[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

**Paradigmi  
computazionali**

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

Esempio 3

Conclusioni

# Paradigmi computazionali



# Paradigmi

[Introduzione al corso](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)

[Paradigmi](#)

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

Esempio 3

Conclusioni

**Imperativo:** Un programma specifica sequenze di modifiche da apportare allo *stato della macchina* (memoria).



# Paradigmi

[Introduzione al corso](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)

[Paradigmi](#)

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

Esempio 3

Conclusioni

**Imperativo:** Un programma specifica sequenze di modifiche da apportare allo *stato della macchina* (memoria).

**Funzionale:** Il programma e le sue componenti sono *funzioni*. Esecuzione come valutazione di funzioni.



# Paradigmi

[Introduzione al corso](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)

[Paradigmi](#)

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

Esempio 3

Conclusioni

**Imperativo:** Un programma specifica sequenze di modifiche da apportare allo *stato della macchina* (memoria).

**Funzionale:** Il programma e le sue componenti sono *funzioni*. Esecuzione come valutazione di funzioni.

**Logico:** Programma come descrizione logica di un problema. Esecuzione analoga a processi di dimostrazione di teoremi.



# Paradigmi

[Introduzione al corso](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)

[Paradigmi](#)

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

Esempio 3

Conclusioni

**Imperativo:** Un programma specifica sequenze di modifiche da apportare allo *stato della macchina* (memoria).

**Funzionale:** Il programma e le sue componenti sono *funzioni*. Esecuzione come valutazione di funzioni.

**Logico:** Programma come descrizione logica di un problema. Esecuzione analoga a processi di dimostrazione di teoremi.

**Orientato ad oggetti:** Programma costituito da oggetti che scambiano messaggi.



# Paradigmi

[Introduzione al corso](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)

[Paradigmi](#)

[Esempio 1.0  
Imperativo vs.  
Funzionale](#)

[Esempio 1.1](#)

[Esempio 1.2](#)

[Esempio 2  
Esempio 2](#)

[Esempio 3  
Esempio 3](#)

[Conclusioni](#)

**Imperativo:** Un programma specifica sequenze di modifiche da apportare allo *stato della macchina* (memoria).

**Funzionale:** Il programma e le sue componenti sono *funzioni*. Esecuzione come valutazione di funzioni.

**Logico:** Programma come descrizione logica di un problema. Esecuzione analoga a processi di dimostrazione di teoremi.

**Orientato ad oggetti:** Programma costituito da oggetti che scambiano messaggi.

**Parallello:** Programmi che descrivono entità distribuite che sono eseguite contemporaneamente ed in modo asincrono.

Gli ultimi due sono ortogonali rispetto ai primi tre...



# Esempio 1.0 Imperativo vs. Funzionale

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

[Paradigmi](#)

**Esempio 1.0  
Imperativo vs.  
Funzionale**

[Esempio 1.1](#)

[Esempio 1.2](#)

[Esempio 2](#)

[Esempio 2](#)

[Esempio 3](#)

[Esempio 3](#)

[Conclusioni](#)

## function definition

```
function factI (n)
    local accumulator = 1
    for i = 1,n do
        accum = accumulator*i
    end
    return accum
end
```

## trace of execution

```
factI(4) :
    accumulator = 1
    i = 1    accumulator = 1 * 1
    i = 2    accumulator = 1 * 2
    i = 3    accumulator = 2 * 3
    i = 4    accumulator = 6 * 4
    return 24
```

## function definition

```
function factR (n)
    if n == 1 then
        return 1
    else
        return n*factR(n-1)
    end
end
```

## trace of execution

```
factR(4) =
4 * factR(3) =
    3 * factR(2) =
        2 * factR(1) =
            1
            1
            1
            2
6
```

24

Notare eliminazione assegnamenti:  $n$  rimpiazza  $i$ , ricorsione invece di cicli



## Esempio 1.1

Vogliamo scrivere in un linguaggio imperativo, funzionale e logico la funzione `member(X, L)` che decida se l'elemento `X` appartiene alla lista `L`.

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

[Paradigmi  
Esempio 1.0  
Imperativo vs.  
Funzionale](#)

[Esempio 1.1](#)

[Esempio 1.2](#)

[Esempio 2](#)

[Esempio 2](#)

[Esempio 3](#)

[Esempio 3](#)

[Conclusioni](#)



## Esempio 1.1

Vogliamo scrivere in un linguaggio imperativo, funzionale e logico la funzione `member(X, L)` che decida se l'elemento `X` appartiene alla lista `L`.

Es.:

```
member(2, [1, 2, 3]) = true;  
member(4, [1, 2, 3]) = false.
```

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

[Paradigmi  
Esempio 1.0  
Imperativo vs.  
Funzionale](#)

**Esempio 1.1**

[Esempio 1.2](#)

[Esempio 2](#)

[Esempio 2](#)

[Esempio 3](#)

[Esempio 3](#)

[Conclusioni](#)



## Esempio 1.1

Vogliamo scrivere in un linguaggio imperativo, funzionale e logico la funzione `member(X, L)` che decida se l'elemento `X` appartiene alla lista `L`.

Es.:

`member(2, [1, 2, 3]) = true;`

`member(4, [1, 2, 3]) = false.`

A questo scopo, si suppongano già esistenti le funzioni:

- `empty(L)`, che restituisce `true` se `L` è vuota, altrimenti `false`.

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

[Paradigmi](#)

[Esempio 1.0](#)

[Imperativo vs.](#)

[Funzionale](#)

**Esempio 1.1**

[Esempio 1.2](#)

[Esempio 2](#)

[Esempio 2](#)

[Esempio 3](#)

[Esempio 3](#)

[Conclusioni](#)



## Esempio 1.1

Vogliamo scrivere in un linguaggio imperativo, funzionale e logico la funzione `member(X, L)` che decida se l'elemento `X` appartiene alla lista `L`.

Es.:

```
member(2, [1, 2, 3]) = true;
```

```
member(4, [1, 2, 3]) = false.
```

A questo scopo, si suppongano già esistenti le funzioni:

- `empty(L)`, che restituisce `true` se `L` è vuota, altrimenti `false`.
- `first(L)`, che restituisce il primo elemento della lista `L`.

Es.: `first([1, 2, 3]) = 1.`

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

[Paradigmi](#)

[Esempio 1.0](#)

[Imperativo vs.](#)

[Funzionale](#)

**Esempio 1.1**

[Esempio 1.2](#)

[Esempio 2](#)

[Esempio 2](#)

[Esempio 3](#)

[Esempio 3](#)

[Conclusioni](#)



## Esempio 1.1

Vogliamo scrivere in un linguaggio imperativo, funzionale e logico la funzione `member(X, L)` che decida se l'elemento `X` appartiene alla lista `L`.

Es.:

`member(2, [1, 2, 3]) = true;`

`member(4, [1, 2, 3]) = false.`

A questo scopo, si suppongano già esistenti le funzioni:

- `empty(L)`, che restituisce `true` se `L` è vuota, altrimenti `false`.
- `first(L)`, che restituisce il primo elemento della lista `L`.  
Es.: `first([1, 2, 3]) = 1.`
- `rest(L)`, che restituisce una sottolista ottenuta rimuovendo il primo elemento di `L`.  
Es.: `rest([1, 2, 3]) = [2, 3].`



## Esempio 1.2

La funzione member nel paradigma imperativo (pseudo-codice):

```
procedure member(X,L)
    local L1 = L
    while not empty(L1) and X != first(L1)
        do L1 = rest(L1)
    return not empty(L1)
```

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi  
Esempio 1.0  
Imperativo vs.

Funzionale

Esempio 1.1

**Esempio 1.2**

Esempio 2

Esempio 2

Esempio 3

Esempio 3

Conclusioni



## Esempio 1.2

La funzione `member` nel paradigma imperativo (pseudo-codice):

```
procedure member(X,L)
    local L1 = L
    while not empty(L1) and X != first(L1)
        do L1 = rest(L1)
    return not empty(L1)
```

In C (un concreto linguaggio imperativo):

```
bool member(int X, Lista L) {
    Lista L1 = L;
    while( ! empty(L1) && X != first(L1))
        L1 = rest(L1);
    return ! empty(L1);
}
```

NB: nessuna differenza strutturale, solo dettagli sintattici



## Esempio 2

La funzione member nel paradigma funzionale:

```
fun member X L =
    if (empty L) then false
    else if X = (first L) then true
    else member X (rest L)
```

[Introduzione al corso](#)

[Linguaggi \(di  
programmazione\)](#)

[Macchine astratte](#)

[Paradigmi  
computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

**Esempio 2**

Esempio 2

Esempio 3

Esempio 3

Conclusioni



## Esempio 2

La funzione member nel paradigma funzionale:

```
fun member X L =  
    if (empty L) then false  
    else if X = (first L) then true  
    else member X (rest L)
```

Nella versione funzionale pura non ci sono variabili, nè assegnazioni.  
Quindi non si possono usare cicli e bisogna rimpiazzarli con la ricorsione.

La sintassi Lisp e Scheme sarebbe un po' particolare:

```
(defun member (x l)  
  (cond ((null l) nil)  
        ((equal x (first l)) T)  
        (T (member x (rest l)))))
```

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

**Esempio 2**

Esempio 2

Esempio 3

Esempio 3

Conclusioni



## Esempio 2

La funzione member nel paradigma funzionale:

```
fun member X L =  
    if (empty L) then false  
    else if X = (first L) then true  
    else member X (rest L)
```

Anche il C si potrebbe usare in stile funzionale se evitassimo di usare i costrutti imperativi:

```
bool member(int X, Lista L) {  
    return (empty(L))?  
            false :  
            (  
                (X == first(L))?  
                true :  
                member(X, rest(L))  
            );  
}
```

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

**Esempio 2**

Esempio 3

Esempio 3

Conclusioni



## Esempio 3

La funzione `member` nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

- `[X|L]` è un *pattern* che denota una lista in cui `X` è la testa e `L` è il resto
- `member` è un *predicato*
- I parametri formali dei predicati possono essere pattern
- I programmi consistono di definizioni di predicati

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

[Paradigmi  
Esempio 1.0](#)

[Imperativo vs.  
Funzionale](#)

[Esempio 1.1](#)

[Esempio 1.2](#)

[Esempio 2](#)

[Esempio 2](#)

[Esempio 3](#)

[Esempio 3](#)

[Conclusioni](#)



## Esempio 3

La funzione `member` nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- `member(2, [1,2,3])` restituisce *yes* (`true`)
- `member(0, [1,2,3])` restituisce *no* (`false`)

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

**Esempio 3**

Conclusioni



## Esempio 3

La funzione `member` nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- `member(2, [1,2,3])` restituisce *yes* (`true`)
- `member(0, [1,2,3])` restituisce *no* (`false`)
- Query con variabili: `member(X,[1,2,3])` restituisce
  - ◆ `X=1`

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi  
Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

**Esempio 3**

Conclusioni



## Esempio 3

La funzione `member` nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- `member(2, [1,2,3])` restituisce *yes* (`true`)
- `member(0, [1,2,3])` restituisce *no* (`false`)
- Query con variabili: `member(X,[1,2,3])` restituisce
  - ◆ `X=1; X=2`

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

**Esempio 3**

Conclusioni



## Esempio 3

La funzione `member` nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- `member(2, [1,2,3])` restituisce *yes* (`true`)
- `member(0, [1,2,3])` restituisce *no* (`false`)
- Query con variabili: `member(X,[1,2,3])` restituisce
  - ◆ `X=1; X=2; X=3`

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

**Esempio 3**

Conclusioni



## Esempio 3

La funzione `member` nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- `member(2, [1,2,3])` restituisce *yes* (`true`)
- `member(0, [1,2,3])` restituisce *no* (`false`)
- Query con variabili: `member(X,[1,2,3])` restituisce
  - ◆ `X=1; X=2; X=3; no` (si comporta come un generatore)

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

**Esempio 3**

Conclusioni



## Esempio 3

La funzione member nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- `member(2, [1,2,3])` restituisce *yes* (`true`)
- `member(0, [1,2,3])` restituisce *no* (`false`)
- Query con variabili: `member(X,[1,2,3])` restituisce
  - ◆ `X=1; X=2; X=3; no` (si comporta come un generatore)
- Risposte con variabili: `member(1,L)` restituisce
  - ◆ `L=[1 | L0]`

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

Esempio 3

Conclusioni



## Esempio 3

La funzione member nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- member(2, [1,2,3]) restituisce *yes* (*true*)
- member(0, [1,2,3]) restituisce *no* (*false*)
- Query con variabili: member(X,[1,2,3]) restituisce
  - ◆ X=1; X=2; X=3; no (si comporta come un generatore)
- Risposte con variabili: member(1,L) restituisce
  - ◆ L=[1|L<sub>0</sub>]; L=[Y<sub>0</sub>,1|L<sub>1</sub>]

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

Esempio 3

Conclusioni



## Esempio 3

La funzione member nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- `member(2, [1,2,3])` restituisce *yes* (`true`)
- `member(0, [1,2,3])` restituisce *no* (`false`)
- Query con variabili: `member(X,[1,2,3])` restituisce
  - ◆ `X=1; X=2; X=3; no` (si comporta come un generatore)
- Risposte con variabili: `member(1,L)` restituisce
  - ◆ `L=[1|L0]; L=[Y0,1|L1]; L=[Y0,Y1,1|L2] ...`

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

**Esempio 3**

Conclusioni



## Esempio 3

La funzione `member` nel paradigma logico (prolog):

```
member(X, [X|L]).  
member(X, [Y|L]) :- member(X, L).
```

Esecuzione:

- `member(2, [1,2,3])` restituisce *yes* (`true`)
- `member(0, [1,2,3])` restituisce *no* (`false`)
- Query con variabili: `member(X,[1,2,3])` restituisce
  - ◆  $X=1; X=2; X=3; \dots$  no (si comporta come un generatore)
- Risposte con variabili: `member(1,L)` restituisce
  - ◆  $L=[1|L_0]; L=[Y_0,1|L_1]; L=[Y_0,Y_1,1|L_2] \dots$
- *Invertibilità*: nessuna distinzione tra input e output. Un solo predicato (programma), molte funzioni.

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

Paradigmi

Esempio 1.0

Imperativo vs.

Funzionale

Esempio 1.1

Esempio 1.2

Esempio 2

Esempio 2

Esempio 3

**Esempio 3**

Conclusioni



# Conclusioni

- Il paradigma di appartenenza può influenzare *radicalmente* il modo in cui si risolve il problema
  - ◆ In linguaggi dello stesso paradigma, lo stesso problema ha soluzioni strutturalmente identiche
  - ◆ Imparato a risolvere un problema in un linguaggio, lo si sa risolvere in tutti i linguaggi dello stesso paradigma
- Il paradigma non è l'unico aspetto determinante. Altri esempi di aspetti importanti:
  - ◆ Il sistema di tipi supportato
  - ◆ Eventuale supporto alle eccezioni
  - ◆ Modello di concorrenza e sincronizzazione
  - ◆ Presenza di garbage collection
  - ◆ ...

[Introduzione al corso](#)

[Linguaggi \(di programmazione\)](#)

[Macchine astratte](#)

[Paradigmi computazionali](#)

[Paradigmi  
Esempio 1.0  
Imperativo vs.  
Funzionale](#)

[Esempio 1.1  
Esempio 1.2](#)

[Esempio 2  
Esempio 2](#)

[Esempio 3  
Esempio 3](#)

[Conclusioni](#)