Fondamenti di Programmazione

Prof. Andrea Sterbini

Corso di studi in Informatica – Primo Anno – Primo Semestre
Università di Roma, La Sapienza

Lez. 2 - 28 settembre 2023

SAPIENZA
UNIVERSITÀ DI ROMA

Fondamenti di programmazione

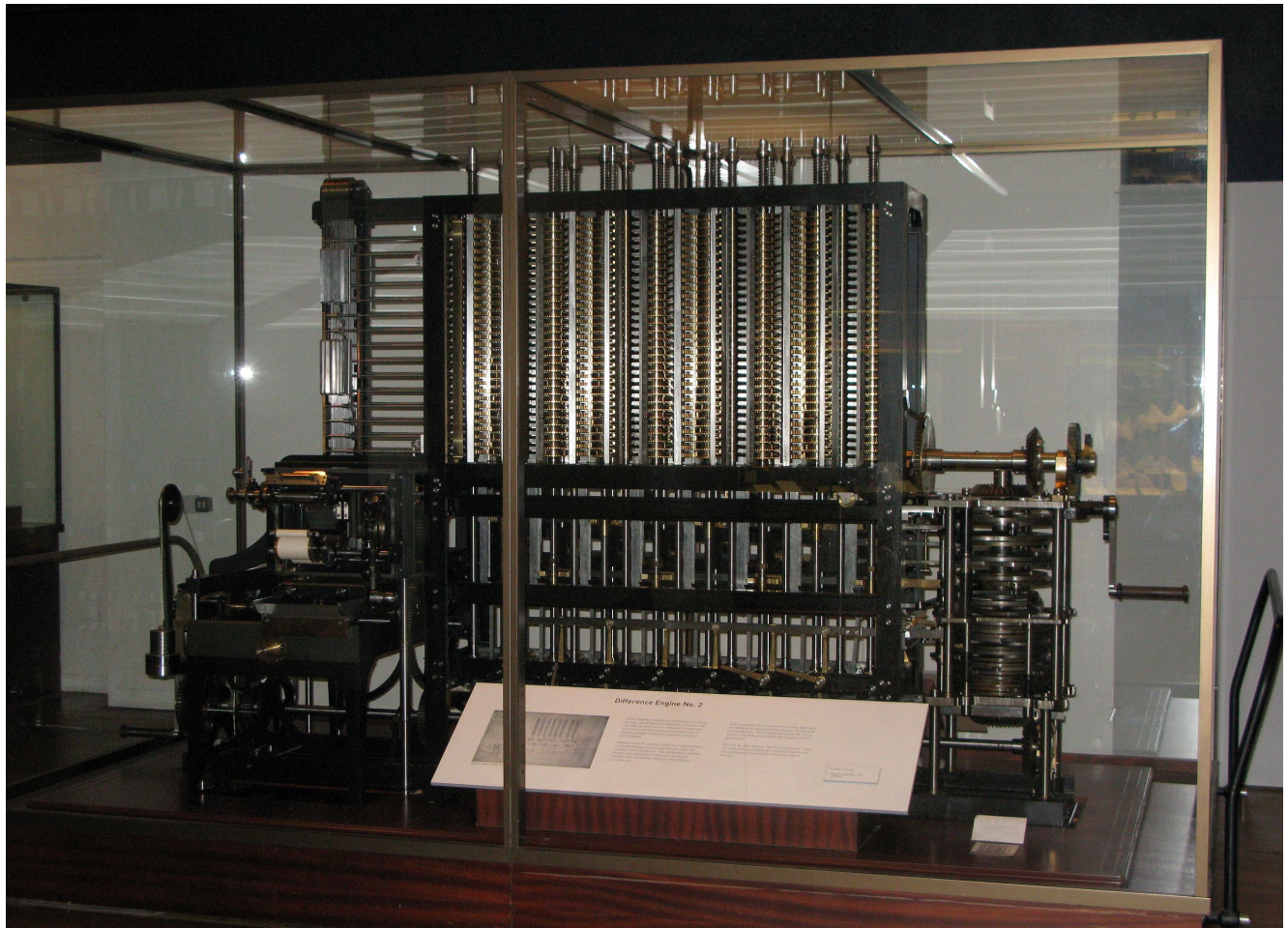
I Computer e la Programmazione

Micro storia dei Computer e dei Linguaggi di programmazione

- 1800: Telai **Jacquard**: con schede perforate si creavano disegni delle stoffe

Micro storia dei Computer e dei Linguaggi di programmazione

- 1800: Telai **Jacquard**: con schede perforate si creavano disegni delle stoffe
- 1822: **Charles Babbage** progetta la "Difference Engine"
  - (che non fu costruita per via della tecnologia dell'epoca)

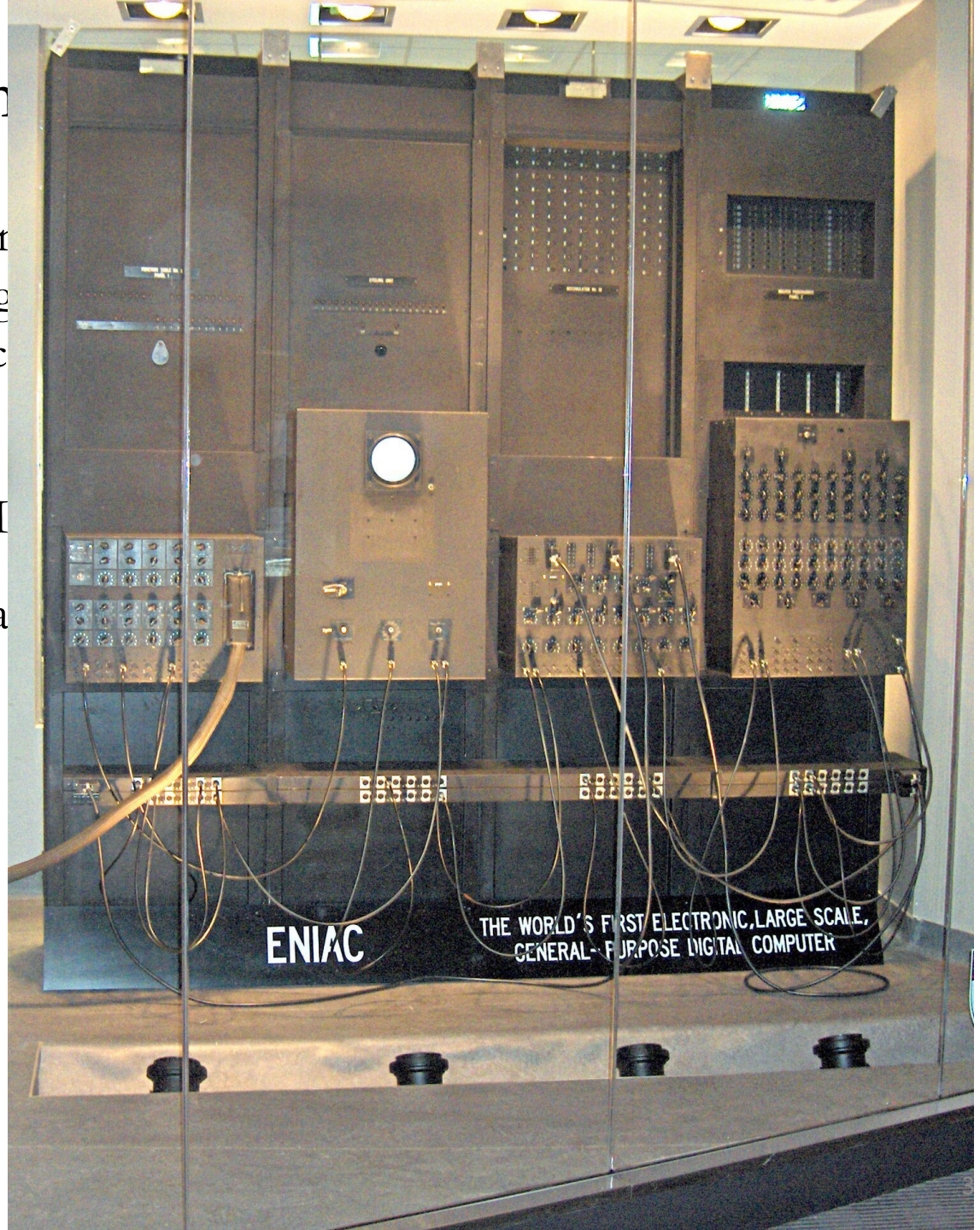


Micro storia dei Computer e dei Linguaggi di programmazione

- 1800: Telai **Jacquard**: con schede perforate si creavano disegni delle stoffe
- 1822: **Charles Babbage** progetta la ”Difference Engine”
  - (che non fu costruita per via della tecnologia dell’epoca)
- 1848: **Ada Lovelace** scrive il primo programma (per la macchina di Babbage)

Micro storia dei Computer e dei Lin

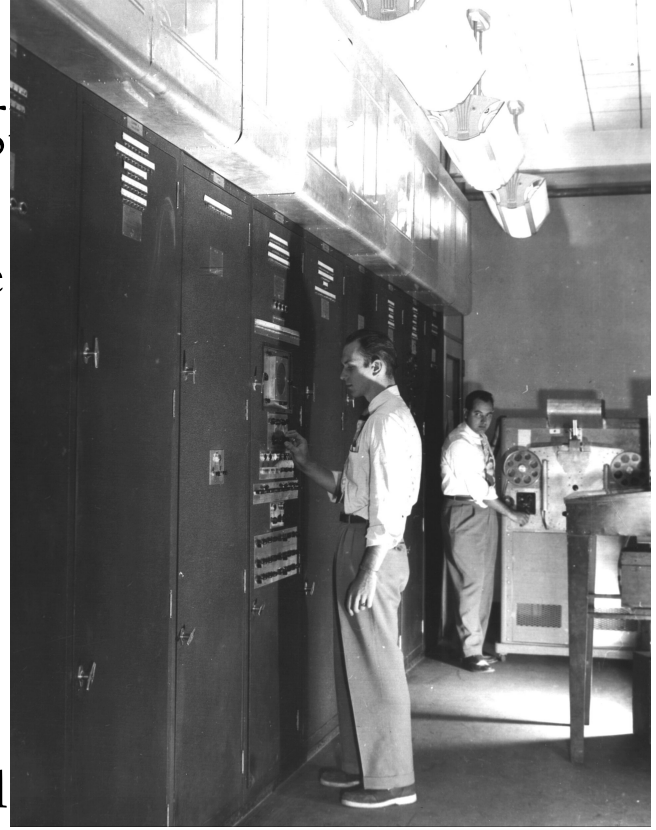
- 1800: Telai **Jacquard**: con schede perforate si creavar
- 1822: **Charles Babbage** progetta la "Difference Eng
  - (che non fu costruita per via della tecnologia dell'epoc
- 1848: **Ada Lovelace** scrive il primo programma (per la macchina di Babbage)
- 1940-46: I primi computer (ENIAC, COLOSSUS, M erano dei grossi circuiti logici da collegare tra loro per realizzare il programma desiderato (logica cablata



Micro storia dei Computer e dei Linguaggi di programmazione

- 1800: Telai **Jacquard**: con schede perforate si creavano disegni delle stoffe
- 1822: **Charles Babbage** progetta la "Difference Engine"
  - (che non fu costruita per via della tecnologia dell'epoca)
- 1848: **Ada Lovelace** scrive il primo programma
  (per la macchina di Babbage)
- 1940-46: I primi computer (ENIAC, COLOSSUS, MARK1, …)
  erano dei grossi circuiti logici da collegare tra loro
  per realizzare il programma desiderato (logica cablata)
- Turing e von Neumann introducono l'architettura del "**computer general purpose**"

Micro storia dei Computer e dei Linguaggi di prog



- 1800: Telai **Jacquard**: con schede perforate si creavano disegni delle stoffe
- 1822: **Charles Babbage** progetta la "Difference Engine"
  - (che non fu costruita per via della tecnologia dell'epoca)
- 1848: **Ada Lovelace** scrive il primo programma
  (per la macchina di Babbage)
- 1940-46: I primi computer (ENIAC, COLOSSUS, MARK1, …)
  erano dei grossi circuiti logici da collegare tra loro
  per realizzare il programma desiderato (logica cablata)
- Turing e von Neumann introducono l'architettura del "**computer general**
- 1948: EDVAC, il primo computer a "**programma memorizzato**" nella memoria
  - Nascono le **istruzioni** che automatizzano la re-configurazione dei circuiti
  - I programmi si scrivevano con le istruzioni del processore (il "**linguaggio macchina**")
  - Gestione "manuale" di tutti gli **indirizzamenti** (salti del controllo e accessi ai dati in memoria)

Micro storia dei Computer e dei Linguaggi di programmazione

```
# esempio in assembly MIPS (che userete ad Architetture degli elaboratori)

# vogliamo calcolare C = A + B
.data                       # definiamo le variabili ed i valori iniziali
A: .word 100                # alloco una word e gli do il nome 'A' e valore 100
B: .word 1                  # alloco una word e gli do il nome 'B' e valore 1
C: .word 0                  # alloco una word e gli do il nome 'C' e valore 0
.text                       # definiamo le istruzioni
main:                       # il programma inizia da questo punto
   lw $t0, A                # carico dalla memoria il valore di A nel registro $t0
   lw $t1, B                # carico B in $t1
   add $t2, $t1, $t0        # sommo $t1 a $t0 e metto il risultato in $t2
   sw $t2, C                # salvo il risultato $t2 in memoria all'indirizzo C
```

  - Gestione "manuale" di tutti gli **indirizzamenti** (salti del controllo e accessi ai dati in memoria)
- 1949: nasce l'**ASSEMBLY**, un linguaggio mnemonico per scrivere le istruzioni
  - (ciascuna istruzione ha una corrispondenza diretta con una istruzione macchina)
  - Automazione dei riferimenti dei salti ed ai dati in memoria (tramite **etichette**)
  - Gestione "manuale" della trasformazione di espressioni complesse in seq. di istruzioni

I primi linguaggi di “alto livello”

- 1957: **FORTRAN** (FORmula TRANslator, dedicato alla programmazione numerica)
  - Automazione della traduzione di **espressioni algebriche** in sequenze di istruzioni e della **compilazione** del programma nel suo complesso

I primi linguaggi di “alto livello”

- 1957: **FORTRAN** (FORmula TRANslator, dedicato alla programmazione numerica)
  - Automazione della traduzione di **espressioni algebriche** in sequenze di istruzioni e della **compilazione** del programma nel suo complesso
- 1959: **LISP** (LISt Processor, dedicato alla programmazione simbolica ed AI)
  - Anche **il programma è una struttura simbolica** che può essere manipolato/generato

I primi linguaggi di “alto livello”

- 1957: **FORTRAN** (FORmula TRANslator, dedicato alla programmazione numerica)
  - Automazione della traduzione di **espressioni algebriche** in sequenze di istruzioni e della **compilazione** del programma nel suo complesso
- 1959: **LISP** (LISt Processor, dedicato alla programmazione simbolica ed AI)
  - Anche **il programma è una struttura simbolica** che può essere manipolato/generato
- 1964: **BASIC**, dedicato all’insegnamento della programmazione
  - Strutture di controllo costruite “manualmente” con **IF** e **GOTO** (“spaghetti coding”)

I primi linguaggi di "alto livello"

- 1957: **FORTRAN** (FORmula TRANslator, dedicato alla programmazione numerica)
  - Automazione della traduzione di **espressioni algebriche** in sequenze di istruzioni e della **compilazione** del programma nel suo complesso
- 1959: **LISP** (LISt Processor, dedicato alla programmazione simbolica ed AI)
  - Anche **il programma è una struttura simbolica** che può essere manipolato/generato
- 1964: **BASIC**, dedicato all'insegnamento della programmazione
  - Strutture di controllo costruite "manualmente" con **IF** e **GOTO** ("spaghetti coding")
- 1970: **PASCAL**, per l'insegnamento della programmazione **strutturata**
  - Eliminazione del GOTO, introduzione di **costrutti di controllo strutturati** (if,for,while,…)

I primi linguaggi di "alto livello"

- 1957: **FORTRAN** (FORmula TRANslator, dedicato alla programmazione numerica)
  - Automazione della traduzione di **espressioni algebriche** in sequenze di istruzioni e della **compilazione** del programma nel suo complesso
- 1959: **LISP** (LISt Processor, dedicato alla programmazione simbolica ed AI)
  - Anche **il programma è una struttura simbolica** che può essere manipolato/generato
- 1964: **BASIC**, dedicato all'insegnamento della programmazione
  - Strutture di controllo costruite "manualmente" con **IF** e **GOTO** ("spaghetti coding")
- 1970: **PASCAL**, per l'insegnamento della programmazione **strutturata**
  - Eliminazione del GOTO, introduzione di **costrutti di controllo strutturati** (if,for,while,…)
- 1972: Smalltalk, C, SQL, ALGOL, SIMULA
  - **Smalltalk**: programmazione ad oggetti, invenzione delle finestre e degli IDE
  - **C**: facile da compilare, efficiente, usato per creare il sistema operativo Unix
  - **SQL**: per definire le strutture dati e le operazioni dei Data-Base

Linguaggi moderni

Linguaggi moderni

- 1983: **C++, Objective C** estensioni ad oggetti del C, compilati

Linguaggi moderni

- 1983: **C++, Objective C** estensioni ad oggetti del C, compilati
- 1987: **Perl** – linguaggio di scripting, per manipolazione di testi, pagine web dinamiche

Linguaggi moderni

- 1983: **C++, Objective C** estensioni ad oggetti del C, compilati
- 1987: **Perl** – linguaggio di scripting, per manipolazione di testi, pagine web dinamiche
- 1990: **Haskell** – linguaggio dedicato alla matematica

Linguaggi moderni

- 1983: **C++, Objective C** estensioni ad oggetti del C, compilati
- 1987: **Perl** – linguaggio di scripting, per manipolazione di testi, pagine web dinamiche
- 1990: **Haskell** – linguaggio dedicato alla matematica
- 1991: **Python** – linguaggio di scripting, focus su **leggibilità e facilità** (MOLTO USATO)

Linguaggi moderni

- 1983: **C++, Objective C** estensioni ad oggetti del C, compilati
- 1987: **Perl** – linguaggio di scripting, per manipolazione di testi, pagine web dinamiche
- 1990: **Haskell** – linguaggio dedicato alla matematica
- 1991: **Python** – linguaggio di scripting, focus su **leggibilità e facilità** (MOLTO USATO)
- 1995: **Java** ad oggetti, introduce l'indipendenza dal processore (con la Java Virtual Machine)
  - **JavaScript**: interpretato, per eseguire codice nel browser del client
  - **PHP**: interpretato, per creare pagine web generate dinamicamente sul server

Linguaggi moderni

- 1983: **C++, Objective C** estensioni ad oggetti del C, compilati
- 1987: **Perl** – linguaggio di scripting, per manipolazione di testi, pagine web dinamiche
- 1990: **Haskell** – linguaggio dedicato alla matematica
- 1991: **Python** – linguaggio di scripting, focus su **leggibilità e facilità** (MOLTO USATO)
- 1995: **Java** ad oggetti, introduce l'indipendenza dal processore (con la Java Virtual Machine)
  - **JavaScript**: interpretato, per eseguire codice nel browser del client
  - **PHP**: interpretato, per creare pagine web generate dinamicamente sul server
- 2000: **C#** - ad oggetti
- 2003: **Scala e Groovy**: compilati ma più facili di Java, girano sulla JVM
- 2009: **Go** – per descrivere più facilmente programmi paralleli e concorrenti
- 2014: **Swift** – nato dall'Objective C, usato in Apple

Linguaggi **compilati** vs **interpretati**

Linguaggi **compilati** vs **interpretati**

Compilazione

- Analisi del testo del programma
- Generazione assembly o formato intermedio
- Generazione codice oggetto

Linking

- Codice oggetto + librerie => eseguibile

Esecuzione

- Caricamento in memoria del file eseguibile
- Si parte da un indirizzo fissato in memoria
- **PRO:** esecuzione estremamente veloce
- **CON:** va ricompilato ad ogni modifica

Linguaggi **compilati** vs **interpretati**

Compilazione

- Analisi del testo del programma
- Generazione assembly o formato intermedio
- Generazione codice oggetto

Linking

- Codice oggetto + librerie => eseguibile

COMPILE-TIME

Esecuzione

- Caricamento in memoria del file eseguibile
- Si parte da un indirizzo fissato in memoria
- **PRO:** esecuzione estremamente veloce
- **CON:** va ricompilato ad ogni modifica

Linguaggi **compilati** vs **interpretati**

Compilazione

- Analisi del testo del programma
- Generazione assembly o formato intermedio
- Generazione codice oggetto

Linking

- Codice oggetto + librerie => eseguibile

COMPILE-TIME

Esecuzione

- Caricamento in memoria del file eseguibile
- Si parte da un indirizzo fissato in memoria
- **PRO:** esecuzione estremamente veloce
- **CON:** va ricompilato ad ogni modifica

RUN-TIME

Linguaggi **compilati** vs **interpretati**

Compilazione

- Analisi del testo del programma
- Generazione assembly o formato intermedio
- Generazione codice oggetto

Linking

- Codice oggetto + librerie => eseguibile

Esecuzione

- Caricamento in memoria del file eseguibile
- Si parte da un indirizzo fissato in memoria
- **PRO:** esecuzione estremamente veloce
- **CON:** va ricompilato ad ogni modifica

Interpretazione

- Analisi del testo del programma
  - Riga per riga
- Esecuzione di funzioni "builtin" corrispondenti alle espressioni trovate
- **PRO:** facile da modificare
- **CON:** è più lento in esecuzione
  - Analisi ripetuta ad ogni partenza

Esempi

Esempi

LINGUAGGI COMPILATI

- **C / C++ / Objective C / Swift**
- **Pascal / Delphi**
- **Rust**

- **Java** (in realtà non compila in istruzioni native del processore ma in istruzioni di un "processore virtuale", la JVM)
  - **Ottimizzazione**: compilazione nativa del codice virtuale

Esempi

LINGUAGGI COMPILATI

- **C / C++ / Objective C / Swift**
- **Pascal / Delphi**
- **Rust**

- **Java** (in realtà non compila in istruzioni native del processore ma in istruzioni di un "processore virtuale", la JVM)
  - **Ottimizzazione**: compilazione nativa del codice virtuale

LINGUAGGI INTERPRETATI

- BASIC
- Javascript
- **Python**
- Perl
- PHP
- **Ottimizzazioni per rendere più veloce**
  - Analisi e generazione di "bytecode" (elimina ripetizione dell'analisi del testo)
  - Analisi del bytecode e generazione di codice nativo (elimina la interpretazione del "bytecode")

Vantaggi / Svantaggi

Vantaggi / Svantaggi

LINGUAGGI COMPILATI

- **PRO:**
  - Efficienti/veloci
  - Eseguibili non modificabili

- **CON:**
  - Più difficili da programmare
  - **Più difficili da imparare**

Vantaggi / Svantaggi

LINGUAGGI COMPILATI

- **PRO:**
  - Efficienti/veloci
  - Eseguibili non modificabili
- **CON:**
  - Più difficili da programmare
  - **Più difficili da imparare**

LINGUAGGI INTERPRETATI

- **PRO:**
  - Facili da modificare
  - Decentemente ottimizzati
  - Ottima leggibilità
  - **Facili da imparare**
  - Molto usati (Python è al 1°-3° posto)
- **CON:**
  - Eseguibili in genere modificabili

Vantaggi / Svantaggi

LINGUAGGI COMPILATI

- **PRO:**
  - Efficienti/veloci
  - Eseguibili non modificabili
- **CON:**
  - Più difficili da programmare
  - **Più difficili da imparare**

LINGUAGGI INTERPRETATI

- **PRO:**
  - Facili da modificare
  - Decentemente ottimizzati
  - Ottima leggibilità
  - **Facili da imparare**
  - Molto usati (Python è al 1°-3° posto)
- **CON:**
  - Eseguibili in genere modificabili

Vantaggi / Svantaggi

LINGUAGGI COMPILATI

- **PRO:**
  - Efficienti/veloci
  - Eseguibili non modificabili
- **CON:**
  - Più difficili da programmare
  - **Più difficili da imparare**

LINGUAGGI INTERPRETATI

- **PRO:**
  - Facili da modificare
  - Decentemente ottimizzati
  - Ottima leggibilità
  - **Facili da imparare**
  - Molto usati (Python è al 1°-3° posto)
- **CON:**
  - Eseguibili in genere modificabili

Software da installare per programmare in Python

Software da installare per programmare in Python

Distribuzione Python **Anaconda**

per

Windows, MacOS, Linux

https://anaconda.com/download

Software da installare per programmare in Python

Ed inoltre le librerie Python usate nei test automatici:

- **radon**
- **ddt**
- **typeguard**
- **pytest-timeout**
- **stopit**

Distribuzione Python **Anaconda**

per

Windows, MacOS, Linux

https://anaconda.com/download

Software da installare per programmare in Python

Ed inoltre le librerie Python usate nei test automatici:

- **radon**
- **ddt**
- **typeguard**
- **pytest-timeout**
- **stopit**

Altre librerie utili per Spyder:

- **spyder-line-profiler**
- **pytest-profiling**
- **spyder-unittest**

Distribuzione Python **Anaconda**

per

Windows, MacOS, Linux

https://anaconda.com/download

Software da installare per programmare in Python

Ed inoltre le librerie Python usate nei test automatici:

- **radon**
- **ddt**
- **typeguard**
- **pytest-timeout**
- **stopit**

Altre librerie utili per Spyder:

- **spyder-line-profiler**
- **pytest-profiling**
- **spyder-unittest**

- Vedi: q2a.di.uniroma1.it

Distribuzione Python **Anaconda**

per

Windows, MacOS, Linux

https://anaconda.com/download