Dati e Programmi

Informatica@SEFA 2018/2019 - Lezione 2

Massimo Lauria <massimo.lauria@uniroma1.it>
http://massimolauria.net/courses/infosefa2018/

Mercoledì, 26 Settembre 2018

Storia e architettura dei calcolatori

Charles Babbage e Ada Lovelace (183x)

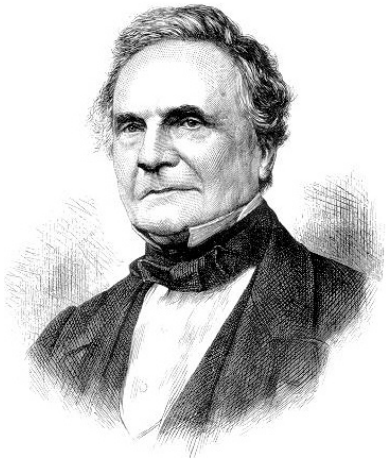


Figure: Chales Babbage (1791–1871)

Figure: Ada Lovelace (1815–1852)

- macchina differenziale (1822)
- macchina analitica (1837) mai realizzata

Alan Turing (1936)

Figure: Alan Turing (1912–1954)

David Hilbert chiede nel 1928: esiste una **procedura meccanica** in grado di stabilire se un'affermazione matematica è un teorema o meno?

Per risolvere il problema si deve spiegare cosa sia una procedura meccanica: la "spiegazione" di Turing è la **macchina di Turing**. Noi le chiamiamo "computer".

La macchina universale

L'idea **fondamentale** della macchina Turing (embrionale in Babbage)

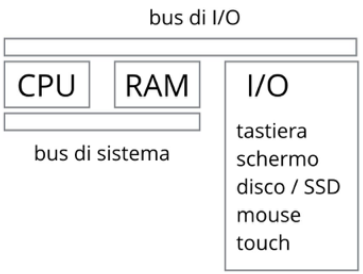
- macchina polivalente e **universale**
- una macchina programmabile
- il programma è un input come gli altri

Tesi di Church-Turing: *tutto ciò che è calcolabile lo è tramite una macchina di Turing più un programma.*

Il risultato è che i computer sono utilizzati per scopi ben oltre quelli per cui sono stati progettati.

Modello di Von Neumann (1945)

Figure: John von Neumann (1903–1957)



Modello su cui sono basati gli odierni calcolatori, più o meno.

- CPU: il cervello
- RAM: la memoria
- I/O: dispositivi di input and output
- bus: canali di comunicazione

Memorie

La memoria RAM è volatile

- tutto viene perduto quando si spegne il computer
- costituita da una gerarchia di memorie (cache L1,L2, memoria principale)

Le memorie secondarie sono permanenti

- dischi rigidi, memorie flash su USB, etc…

Si lavora su RAM ma si salva su memoria secondaria

Tempi di accesso (approssimati, ca. 2010)

Un'istruzione CPU: 1/1.000.000.000 sec = 1 nanosec

Accesso ai dati	tempo (in ns)
memoria cache L1	0.5
memoria cache L2	7
RAM	100
1MB seq. da RAM	250.000
4K random disco rigido SSD	150.000
1MB seq. da disco SSD	1.000.000
disco rigido HDD	8.000.000
1MB seq. da disco HDD	20.000.000
pacchetto dati Europa -> US -> Europa	150.000.000

Fonte: Peter Norvig, `http://norvig.com/21-days.html`

Updated by: Jeff Dean `https://ai.google/research/people/jeff`

Efficienza

Un programma complesso, eseguito su molti dati, può risultare lento.

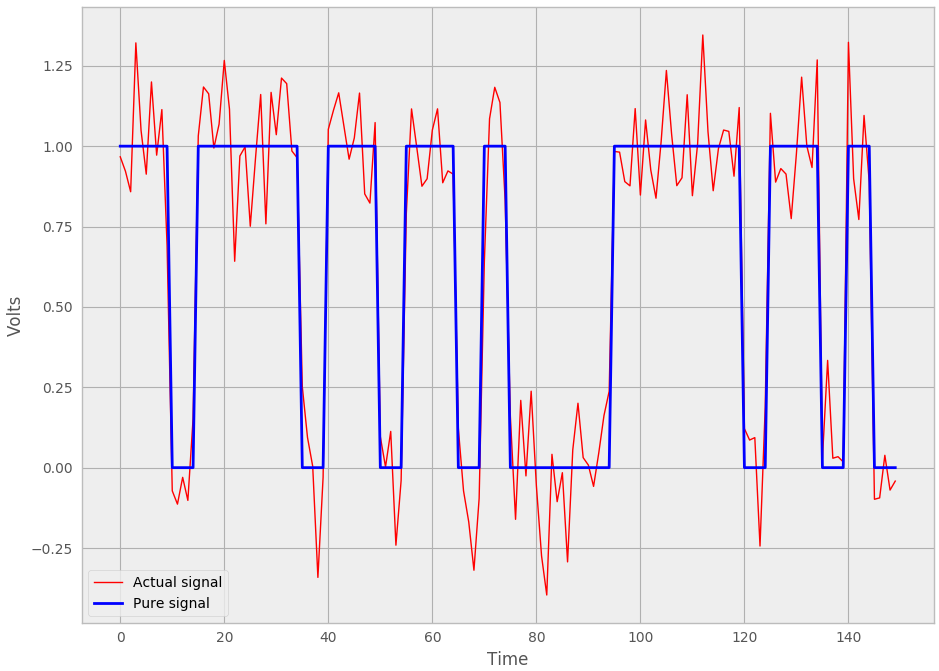
- la correttezza è la cosa principale
- ma un programma troppo lento è inutilizzabile

Dovete ancora imparare a scrivere programmi corretti ed è presto per parlarvi di efficienza, tuttavia

- i vostri programmi devono girare in tempi ragionevoli,
- se non lo fanno probabilmente ci sono errori logici.

Rappresentazione dei dati

Zero e uno



Un **bit di informazione**

- 0 o 1
- Vero o Falso
- tensione alta o bassa
- interruttore acceso o spento

Più di due stati?

- e.g. macchine di Turing
- maggiori errori

Due stati sono pochi: insiemi di bit

n bit assumono 2^n configurazioni

1 byte	8 bit
1 KB	2^10 = 1024 byte
1 MB	2^10 = 1024 KB
1 GB	2^10 = 1024 MB
…	…

Codifica di dati

Il significato di una sequenza di bit dipende dalla sua **interpretazione**

Utilizzare due interpretazioni differenti per gli stessi dati

- può causare errori nel programma
- può corrompere i dati
- può essere usato per violare la sicurezza[1]

[1] Vedremo un esempio di violazione di sicurezza in SQL.

In Python ogni dato ha un **tipo**

```
type(5)        # il tipo dell'espressione 5                    1
type('ciao')   # il tipo dell'espressione 'ciao'               2
type(3.2)      # il tipo dell'espressione 3.2                  3
type(5.0)      # il tipo dell'espressione 5.0                  4
3.2 + 5        # somma tra dati di tipo diverso                5
type(3.2 + 5) # il tipo del risultato                         6
5 + 'ciao'     # altra somma tra dati di tipo diverso          7
```

```
<class 'int'>                                                  1
<class 'str'>                                                  2
<class 'float'>                                                3
<class 'float'>                                                4
8.2                                                            5
<class 'float'>                                                6
Traceback (most recent call last):                             7
  File "<stdin>", line 1, in <module>                          8
TypeError: unsupported operand type(s) for +: 'int' and 'str' 9
```

Il tipo tiene traccia della giusta interpretazione, e aiuta il programmatore a non confondere tipi di dati diversi.

Codifica di numeri

La notazione decimale (i.e. in base 10) usa cifre da 0 a 9

E.g. $45903 = 4 \cdot 10^4 + 5 \cdot 10^3 + 9 \cdot 10^2 + 0 \cdot 10^1 + 3 \cdot 10^0$

Se ci sono solo le cifre 0 e 1 ha senso usare la base 2

E.g. $110101 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

Notazione binaria (E.g. 4 bit)

$$b_3 b_2 b_1 b_0 = b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

b_3 è il bit **più significativo**; b_0 è il bit **meno significativo**

8421			8421		
0000	0	0	1000	8	8
0001	1	1	1001	8+1	9
0010	2	2	1010	8+2	10
0011	2+1	3	1011	8+2+1	11
0100	4	4	1100	8+4	12
0101	4+1	5	1101	8+4+1	13
0110	4+2	6	1110	8+4+2	14
0111	4+2+1	7	1111	8+4+2+1	15

Notazione binaria (n bit)

Caso n bit: $b_{n-1} \dots b_0 = \sum_{i=0}^{n-1} b_i \cdot 2^i$.

8 bit rappresentano $2^8 = 256$ valori

$$\{0, 1, 2, \dots, 255\} \tag{1}$$

16 bit rappresentano $2^{16} = 65536$ valori

$$\{0, 1, 2, \dots, 65535\} \tag{2}$$

Domanda: ogni intero positivo ha esattamente una rappresentazione binaria?

Stampare in notazione binaria

Potete usare Python per fare le conversioni

```
# l'operatore ** è elevazione a potenza                    1
numero = 2**10 + 2**7 + 2**5 + 2**3 + 1                    2
                                                           3
print("Base 10 - {}".format(numero))                       4
print("Base  2 - {:b}".format(numero))                     5
```

```
Base 10 - 1193                                             1
Base  2 - 10010101001                                      2
```

Notazione binaria è scomoda

(dec.) 123456789 = (bin.) 111010110111100110100010101

Base 2

- adatta per il computer
- numeri troppo lunghi e scomodi da leggere

Base 10

- più compatta e leggibile
- le cifre corrispondono male con quelle in base 2

Abbreviazione della notazione binaria

Base 16 (notazione esadecimale)

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

con A = 10, B = 11, C = 12, D = 13, E = 14, F = 15

$$\underbrace{0111}_{7}\ \underbrace{0101}_{5}\ \underbrace{1011}_{B}\ \underbrace{1100}_{C}\ \underbrace{1101}_{D}\ \underbrace{0001}_{1}\ \underbrace{0101}_{5}$$

(dec.) 123456789 = (hex.) 75BCD15

(dec.) 175 = (bin.) 10101111 = (hex.) AF

Usiamo python per verificare i conti

$$\underbrace{0111}_{7}\ \underbrace{0101}_{5}\ \underbrace{1011}_{B}\ \underbrace{1100}_{C}\ \underbrace{1101}_{D}\ \underbrace{0001}_{1}\ \underbrace{0101}_{5}$$

(dec.) 123456789 = (hex.) 75BCD15

```
numero = 123456789                                    1
                                                      2
print("Base 10 - {}".format(numero))                  3
print("Base  2 - {:b}".format(numero))                4
print("Base 16 - {:x}".format(numero))                5
```

```
Base 10 - 123456789                                   1
Base  2 - 111010110111100110100010101                 2
Base 16 - 75bcd15                                     3
```

Rappresentazione di byte

È scomodo scrivere/leggere 8 bit per esteso.
Normalmente un byte è scritto come

- un numero da 0 a 255, oppure
- un numero esadecimale di due cifre, da 00 a FF.

I numeri esadecimali in Python

In python i numeri esadecimali si scrivono col prefisso `0x`

```
(0xFe, 0xfE, 0xfe)                                    1
0XC3                                                  2
0x2D3Ac463e  + 4524                                   3
```

```
(254, 254, 254)                                       1
195                                                   2
12141221866                                           3
```

Ricapitolando

- I numeri codificati come sequenze di 0 e 1…
- …seguendo la notazione in base 2
- Base 16 come abbreviazione di base 2.

Python e numeri molto grandi

```
3**312 + 7**94
```

```
1
```

```
727497445223752651252062953179643967 25
533432866824952575839903695438525721 60
399072891328213522972592220895670826 14
192593410945933875935888274355622 90
```

- Python permette numeri grandi a piacere
- CPU opera su numeri di taglia fissa, e.g. 64 bit
- C, C++, Java, .. fanno lo stesso
- al massimo 2^{64} valori, da -2^{63} a $2^{63}-1$
- possibilità di *overflow*

Codifica della comunicazione scritta

Studente: ‘Prof. una proroga?’

‘Non se ne parla nemmeno.’

Studente: ‘Per favoooore!’

‘Assolutamente no.’

Studente: ‘Che bastardo…’

‘Come?’

Studente: ‘Oops! Sbagliato finestra…’

Codifica della comunicazione scritta

Studente: [80, 114, 111, 102, 46, 32, 117, 110, 97, 32,…]

[78, 111, 110, 32, 115, 101, 32, 110, 101, 32, 112, 97, …]

Studente: [80, 101, 114, 32, 102, 97, 118, 111, 111, 111, 111, 114, 101, 33]

[65, 115, 115, 111, 108, 117, 116, 97, 109, 101, 110, 116, 101, 32, 110, 111, 46]

Studente: [67, 104, 101, 32, 98, 97, 115, 116, 97, 114, 100, 111, 46, 46, 46]

[67, 111, 109, 101, 63]

Studente: [79, 111, 112, 115, 33, 32, 83, 98, 97, 103, 108, 105, 97, …]

Codifica di testi — ASCII

1 byte = $0.b_6b_5b_4.b_3b_2b_1b_0$

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!		#	$	%	&		(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Figure: Tabella ASCII (fonte:Wikipedia)

“Informatica” = (73, 110, 102, 111, 114, 109, 97, 116, 105, 99, 97)

Codifica di testi — ASCII Estesi

La codifica ASCII usa 8 bit ma prevede solo 128 valori. Il bit più significativo è sempre 0.

Esistono varie estensioni di ASCII (e.g. Latin-1)

- usano le stringhe $1b_6b_5b_4b_3b_2b_1b_0$
- caratteri locali e/o con accenti (e.g. à, è, é, ì, ò, ù)
- incompatibili tra loro

Codifica di testi — Unicode e UTF-8

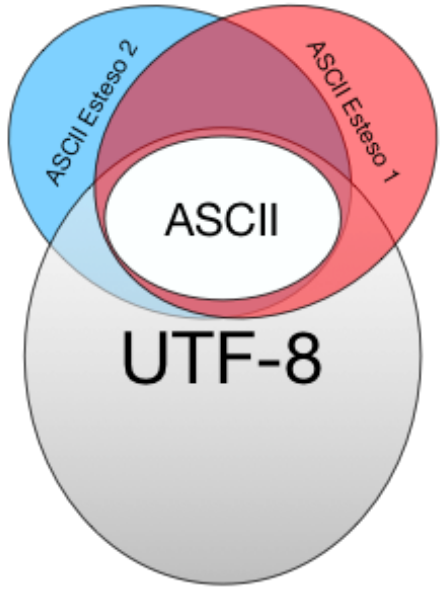
Unicode: una tabella per tutte le lingue

- 32 bit per simbolo = ca. 4 miliardi di simboli
- facilita la comunicazione tra lingue diverse
- supporto per scrittura bidirezionale
- per i testi ASCII è più costosa

UTF-8 è una **rappresentazione** di Unicode

- lunghezza variabile da 1 a 6 byte
- identica ad ASCII per i primi 128 simboli
- lo standard attuale

Codifica di testi — interpretazione ambigua



Esempio 1: ’Caipiriña’ viene codificato in Latin-1

[67, 97, 105, 112, 105, 114, 105, **241**, 97]

il ricevitore decodifica in UTF-8 e segnala errore.

Esempio 2: ’Caipiriña’ viene codificato in UTF-8

[67, 97, 105, 112, 105, 114, 105, **195**, **177**, 97]

il ricevitore decodifica in Latin-1 ’CaipiriÃ±a’

Immagini - bitmap

Bitmap: Griglia di “pixel” colorati con coordinate (x, y)

$$\begin{bmatrix} 0,0 & 1,0 & 2,0 & 3,0 & 4,0 \\ 0,1 & 1,1 & 2,1 & 3,1 & 4,1 \\ 0,2 & 1,2 & 2,2 & 3,2 & 4,2 \\ 0,3 & 1,3 & 2,3 & 3,3 & 4,3 \\ 0,4 & 1,4 & 2,4 & 3,4 & 4,4 \\ 0,5 & 1,5 & 2,5 & 3,5 & 4,5 \\ 0,6 & 1,6 & 2,6 & 3,6 & 4,6 \end{bmatrix}$$

Risoluzione della griglia

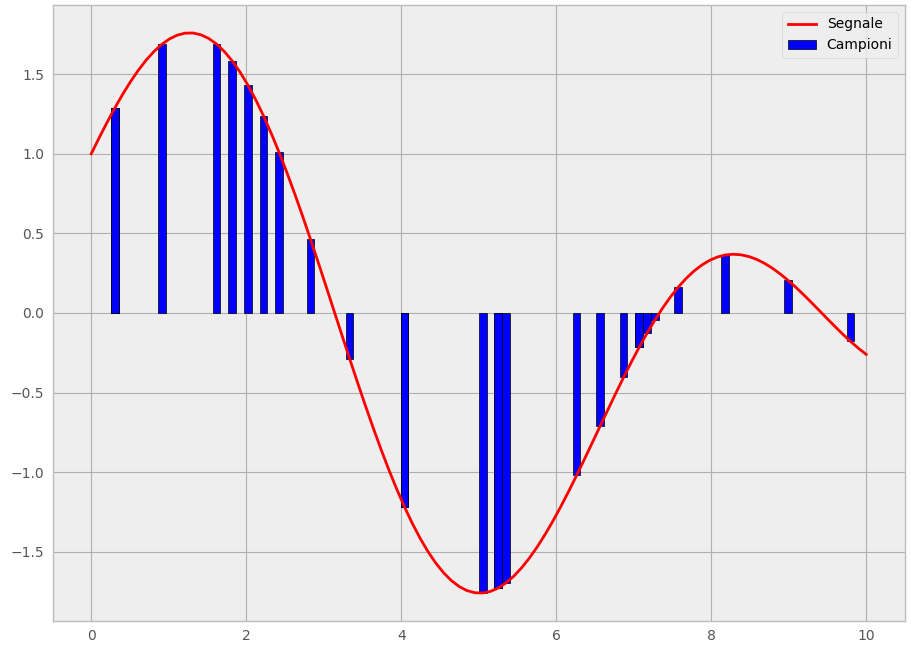
Dimensione immagine =

$$\text{Altezza} \times \text{Larghezza} \times \text{byte(colore)}$$

Spazio:

- Bianco e nero: 1 bit per pixel
- R,G,B : tipicamente 3 byte per pixel
- Tavolozza: $\log_2$(#colori) per pixel + Dim(tavolozza)

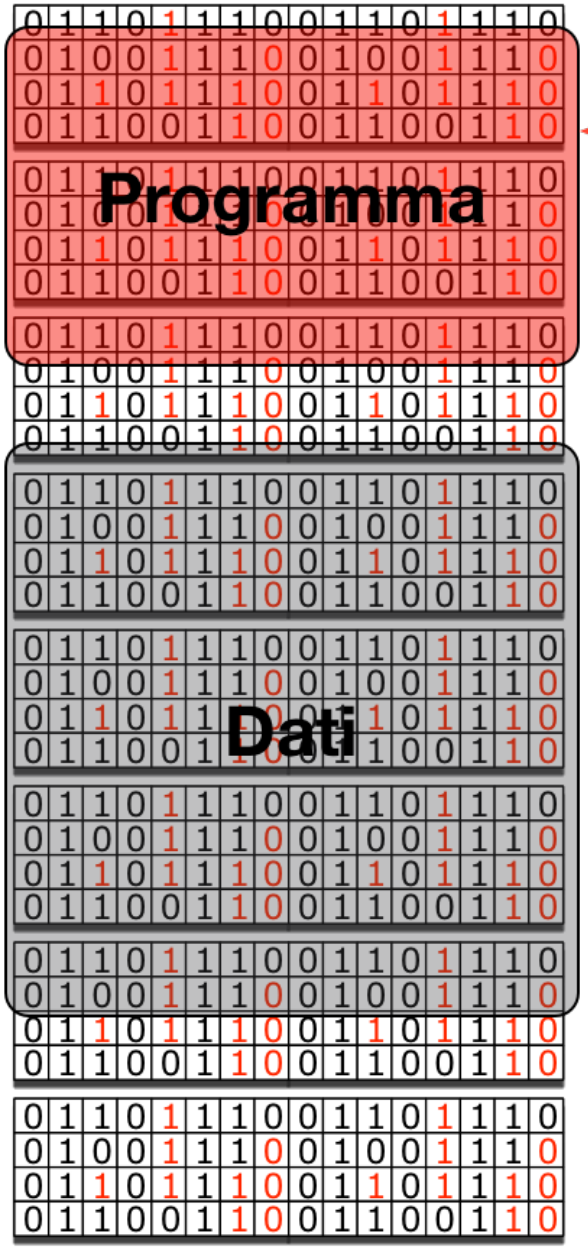
Codifica di segnali: musica, video, forme, …

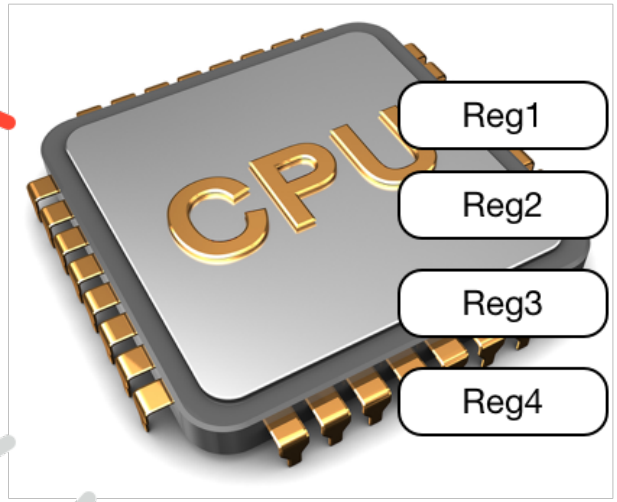


- discretizzazione
- risoluzione
- precisione vs costo
- compressione

Programmazione dei computer

Linguaggio macchina per CPU





1. RAM <-> Registri
   4 o 8 byte
2. Registri <-> Registri

3. Operazioni matematiche
   o logiche tra registri

4. saltare nel programma

Osservazione

Il programma è finito, ma può lavorare su quantità di dati potenzialmente infinita. Questo è possibile grazie alle **istruzioni di salto** della CPU. Ad esempio

Salto assoluto

- E.g., salta alla pos. 531 del programma

Salto condizionato

- E.g., salta alla pos. 421 se il terzo registro è 0

Linguaggi di programmazione evoluti

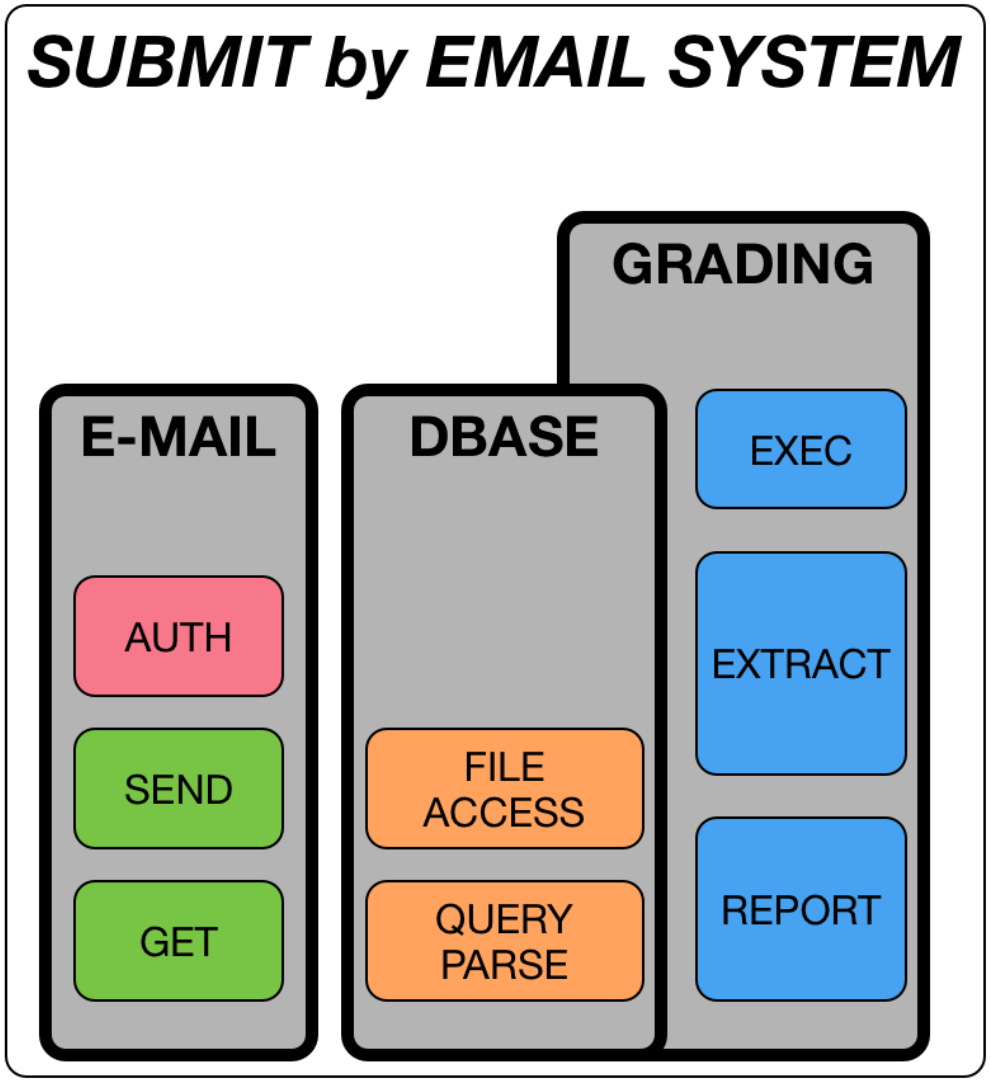
Vogliamo programmare così

```
for x in [1,2,3,4,5]:                                    1
    print(x)                                             2
```

invece di programmare in linguaggio macchina

```
entrypoint:                                              1
        movq    %rdi, -8(%rbp)                           2
        movq    -8(%rbp), %rdi                           3
        cmpq    $0, 80(%rdi)                             4
        sete    %al                                      5
label2:                                                  6
        xorb    $-1, %al                                 7
        andb    $1, %al                                  8
        movzbl  %al, %ecx                                9
        movslq  %ecx, %rdi                               10
```

Astrazione e sotto-problemi



- gerarchia organizzativa
- sotto-problemi e sotto-programmi
- nascondere dettagli
- interfacce
- facile da analizzare
- divisione del lavoro

Astrazioni e organizzazione del pensiero

Le astrazioni sono dei **pezzi logici** che modellano elementi del problema analizzato.
Sono gradini per costruire astrazioni di livello più alto.

Strumenti per le astrazioni

- **Sistema operativo**: dispositivi di I/O, multiprocessi
- **Librerie** (libraries): sotto programmi altrui
- **Elementi del linguaggio**: costruire le proprie astrazioni

Linguaggi di alto e basso livello

Script > L. Applicazioni > L. di Sistema > L. Macchina

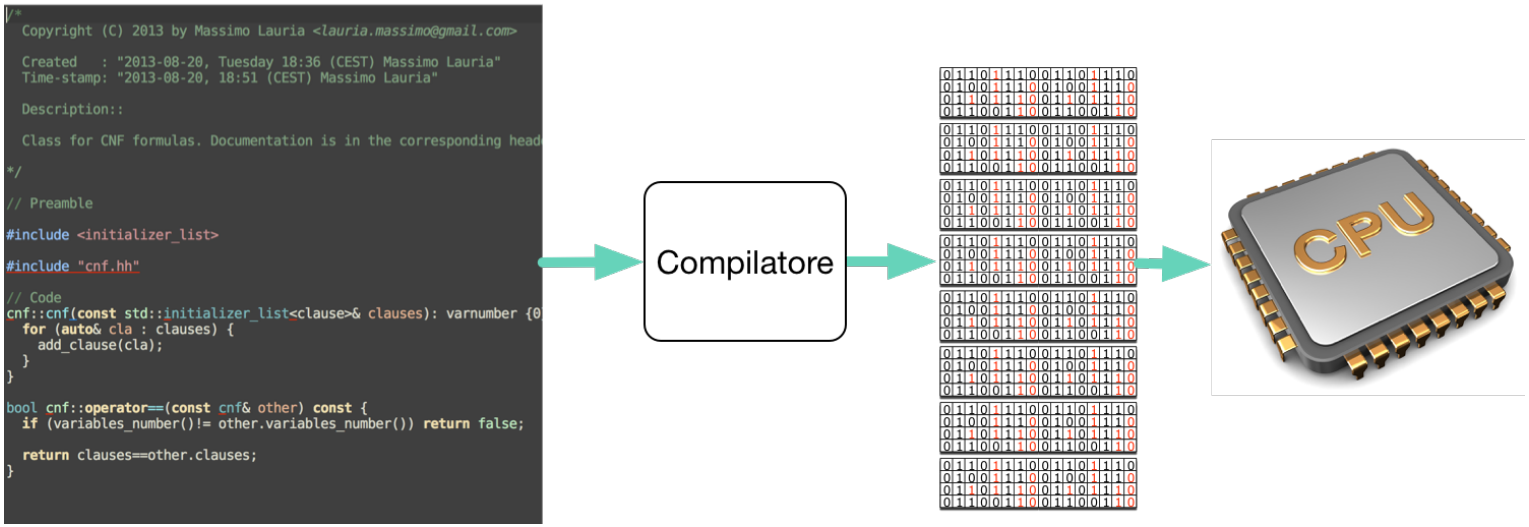
Alto livello

- astrazioni più potenti/espressive
- più facili
- meno efficienti

Basso livello

- astrazioni meno potenti
- più difficili
- più efficienti

Traduzione in blocco (e.g. C, C++)

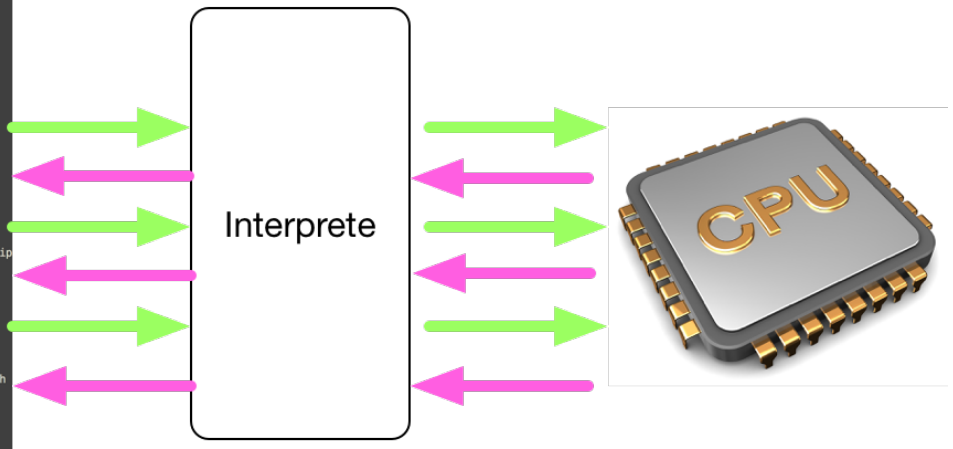


Il programma viene tradotto/ottimizzato in linguaggio macchina, da un **compilatore**, pronto per essere eseguito dalla CPU

- più sicuri
- più efficienti
- meno flessibili
- ling. di alto e basso livello

Esecuzione interattiva (e.g. Python)

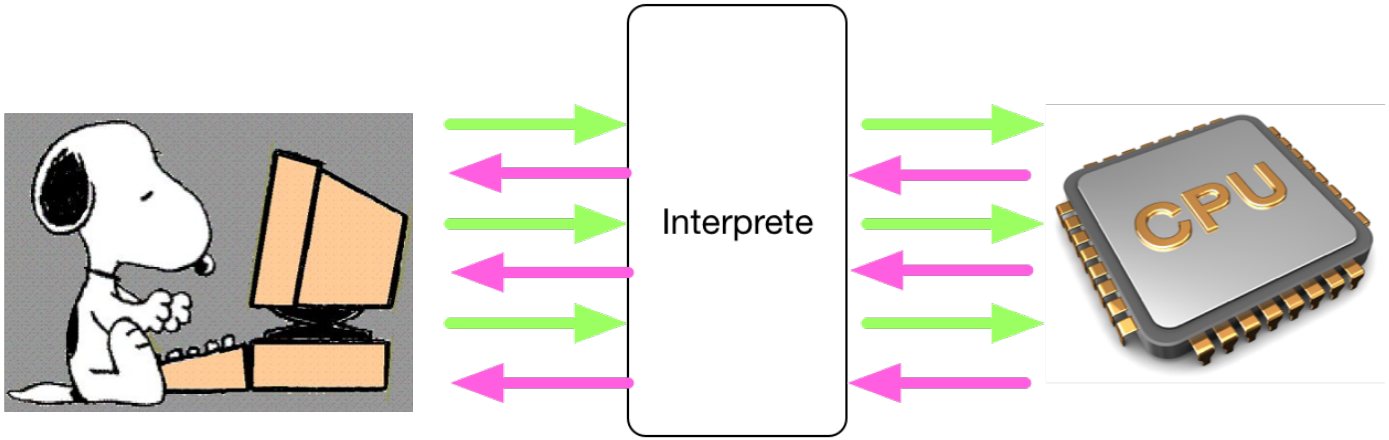
```
r"""Components for command line interface

CNFgen has many command line entry points to its functionality, and
some of them expose the same functionality over and over. This module
contains useful common components.

Copyright (C) 2012, 2013, 2014, 2015, 2016  Massimo Lauria <lauria@kth.se>
https://github.com/MassimoLauria/cnfgen.git

"""

from __future__ import print_function


import sys
import argparse

import networkx
import random

from itertools import combinations, product

from .graphs import supported_formats as graph_formats
from .graphs import readGraph,writeGraph
from .graphs import bipartite_random_left_regular,bipartite_random_regular,bip
ft
from .graphs import bipartite_sets
from .graphs import dag_complete_binary_tree,dag_pyramid
from .graphs import sample_missing_edges


try: # NetworkX >= 1.10

    complete_bipartite_graph    = networkx.bipartite.complete_bipartite_graph
    bipartite_random_graph      = networkx.bipartite.random_graph
    bipartite_gnmk_random_graph = networkx.bipartite.gnmk_random_graph

except AttributeError: # Networkx < 1.10

    from networkx import complete_bipartite_graph
    from networkx import bipartite_random_graph
    from networkx import bipartite_gnmk_random_graph


__all__ = [ "register_cnfgen_subcommand","is_cnfgen_subcommand",
            "DirectedAcyclicGraphHelper", "SimpleGraphHelper", "BipartiteGraph
```



Il programma viene letto da un **interprete** che esegue passo passo quello che è scritto nel programma.

- meno sicuri
- meno efficienti
- più flessibili
- ling. di alto livello

Esecuzione interattiva (e.g. Python)



Il programma viene letto da un **interprete** che esegue passo passo quello che è scritto nel programma.

- meno sicuri
- meno efficienti
- più flessibili
- ling. di alto livello

Python – presentazione ufficiale

```
A = [1,2,3,4,5,6,7,8,9,10]                                    1
B = [ x*x for x in A ]                                        2
print(B)                                                      3
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Python è un linguaggio ad alto livello

- semplice
- libreria molto ricca di funzioni
- interattivo
- più lento di molti altri linguaggi

Python di alto livello (e.g., i numeri)

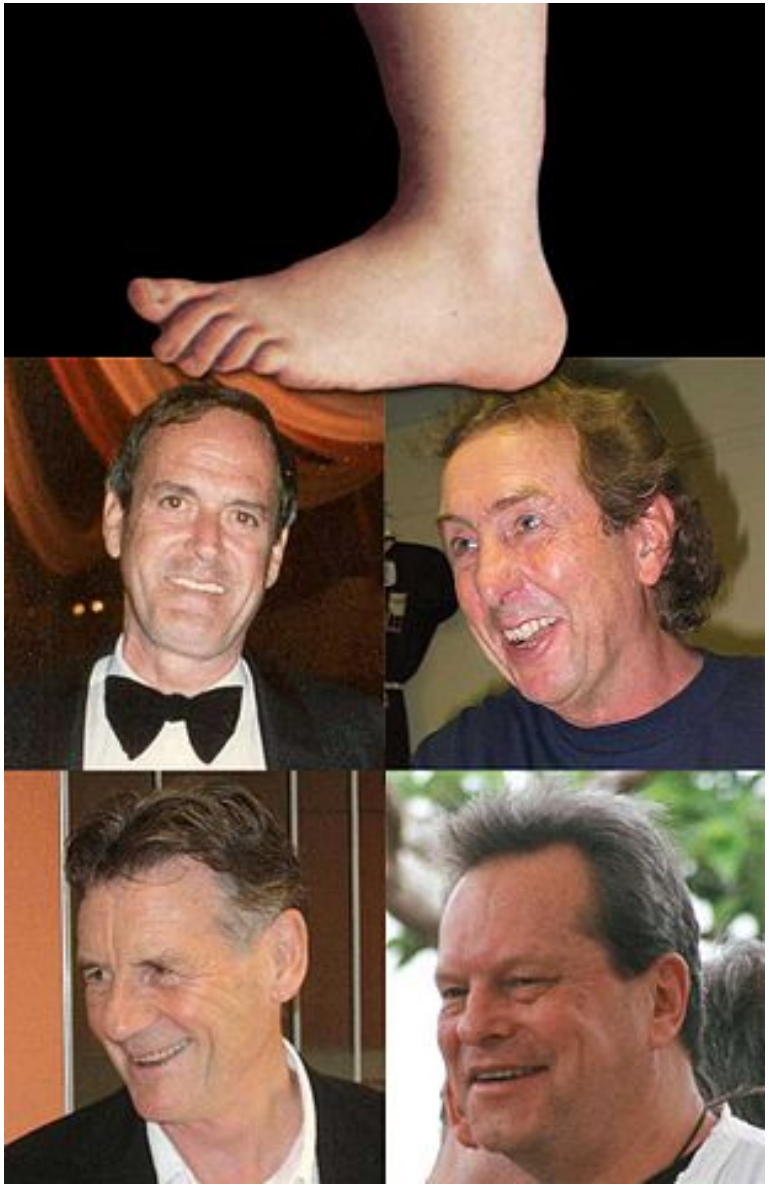
Per esempio Python ha numeri di dimensione arbitraria

- nasconde i dettagli della CPU
- gestisce gli *overflow*

Più ad alto livello di C,C++

- stessi numeri della CPU
- incompatibilità su CPU diverse

Python - risorse (in inglese)



Link utili:

- `http://www.pythontutor.com/`
- `https://docs.python.org/3/`

Strumenti:

- IPython `https://ipython.org/`
- Anaconda: `https://www.anaconda.com/`
- Thonny (offline): `http://thonny.org/`
- repl.it (online): `https://repl.it/`

SQL e basi di dati

```
select ID,name,surname from students                    1
^^I^^I where enroll='2017'                              2
```

ID	name	surname
10231	Mario	Rossi
01234	Giancarlo	Garibaldi
02135	Grace	Hopper
02107	Guybrush	Threepwood
12042	Robert	Wyatt

- richieste dati
- dichiarativo
- standard