

Codifica dei dati

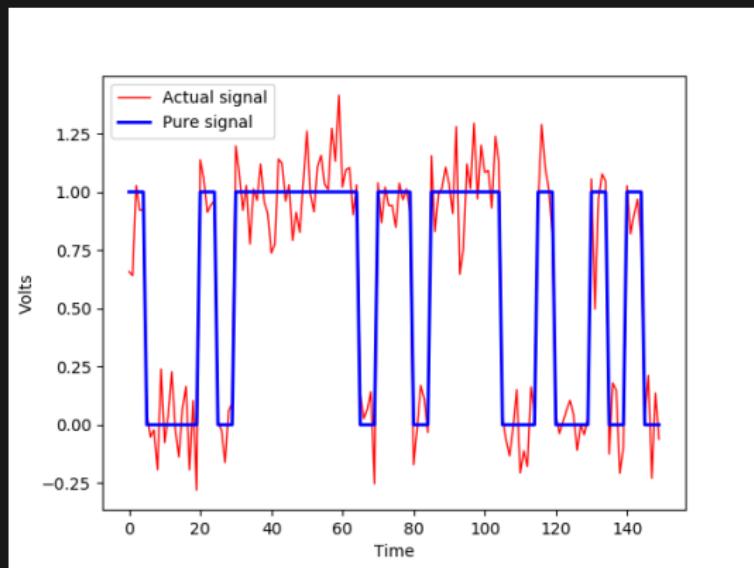
Informatica@DSS 2025/2026

Massimo Lauria <massimo.lauria@uniroma1.it>
<https://massimolauria.net/informatica2025/>

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} KB = 1024 KB
1 GB	2^{10} MB = 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

In Python ogni dato ha un **tipo**

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

```
<class 'int'>
<class 'str'>
<class 'float'>
<class 'float'>
8.2
<class 'float'>
```

Codifica di numeri

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

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

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

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

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

$$\{0, 1, 2, \dots, 255\} \quad (1)$$

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

$$\{0, 1, 2, \dots, 65535\} \quad (2)$$

Domanda: ogni intero positivo ha esattamente una rappresentazione binaria?

Stampare in notazione binaria

Potete usare Python per fare le conversioni

```
# l'operatore ** esegue l'elevazione a potenza      1
numero = 2**10 + 2**7 + 2**5 + 2**3 + 1           2

print("Base 10 - {}".format(numero))                3
print("Base 2 - {:b}".format(numero))                 4
                                                5
```

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

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

0111 0101 1011 1100 1101 0001 0101
7 5 B C D 1 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
Base 2 - 111010110111100110100010101
Base 16 - 75bcd15
```

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 gli esadecimali si scrivono col prefisso 0x

```
print(0xFe, 0xfE, 0xfe)  
print(0XC3)  
print(0x2D3Ac463e + 4524)
```

1
2
3

```
254 254 254  
195  
12141221866
```

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

```
print(3**312 + 7**94)
```

1

```
72749744522375265125206295317964396725
53343286682495257583990369543852572160
39907289132821352297259222089567082614
19259341094593387593588827435562290
```

- ▶ 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

Binario: $0b_6b_5b_4b_3b_2b_1b_0$

Esadecimale: h_1h_0 con $0 \leq h_1 \leq 7$.

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

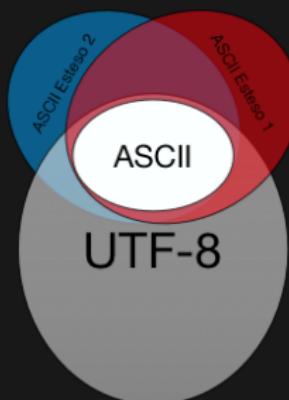
Unicode: una tabella per tutte le lingue

- ▶ 21 bit per simbolo = ca. 2 milioni
- ▶ 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 4 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)



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

Risoluzione della griglia



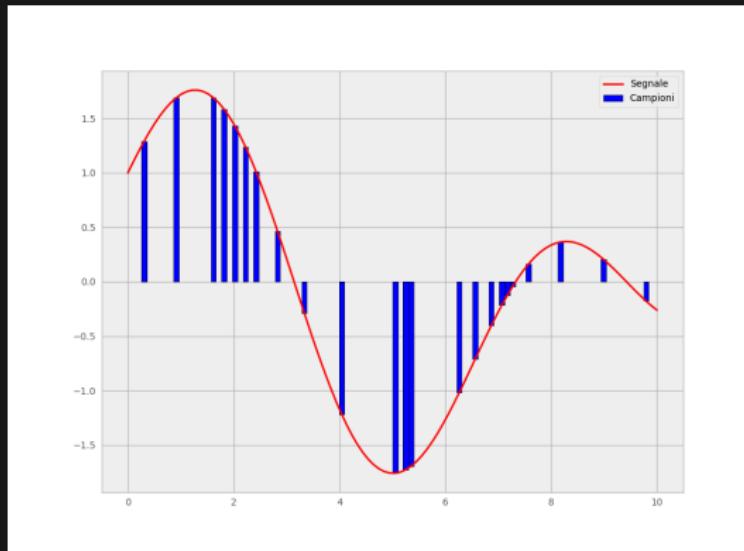
Dimensione immagine =

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

Spazio:

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

Codifica di segnali: musica, video, ...



- discretizzazione
- risoluzione
- precisione vs costo
- compressione

Esercizio

Decodificate il testo seguente, codificato in ASCII

[76, 101, 103, 103, 101, 116, 101, 32, 105, 108, 32, 109, 97, 116, 101, 114, 105, 97, 108, 101, 32, 80, 82, 73, 77, 65, 32, 100, 105, 32, 118, 101, 110, 105, 114, 101, 32, 97, 32, 108, 101, 122, 105, 111, 110, 101, 33]