Corso di Laurea Magistrale in Medicina e Chirurgia – Tecnologie Digitali

Appunti del corso di

"Informatica e elementi di informatica medica"

Prof. Vittorio Lumare
Prof. Pietro Battigaglia

1. Indice

1

1. “Sistemi di numerazione e unità di misura”

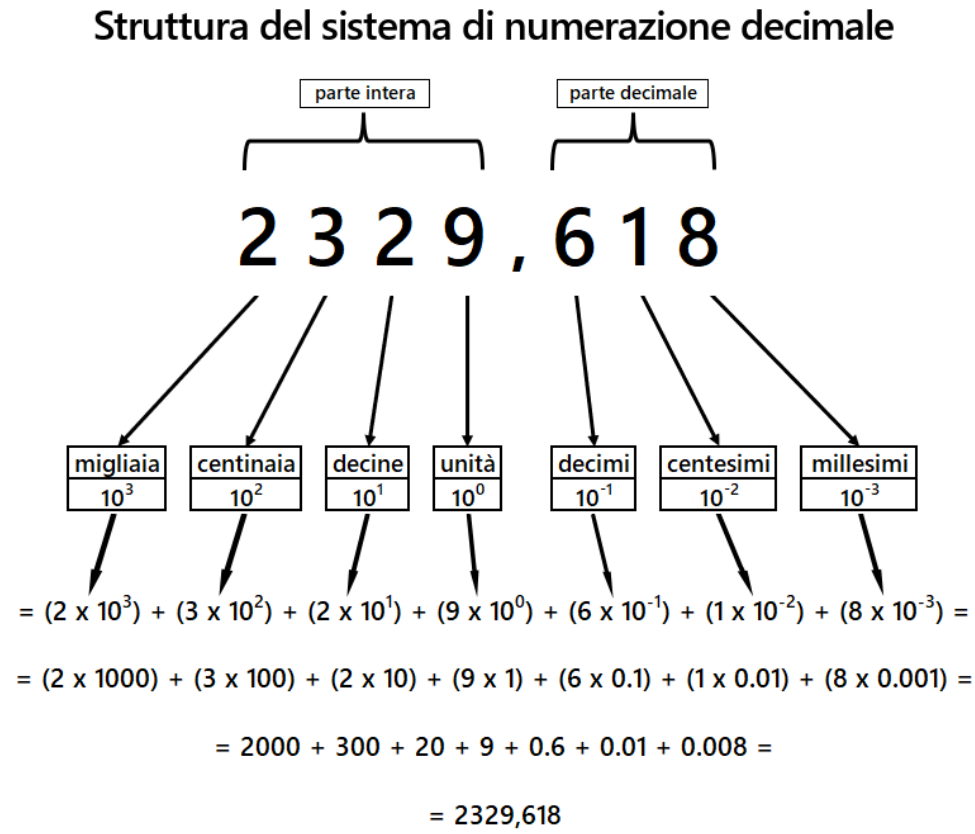
Sistemi di numerazione posizionali

Un sistema di numerazione è un modo di esprimere e rappresentare i numeri attraverso un insieme di simboli. I numeri, fin dai tempi antichi, sono uno strumento necessario per quantificare un insieme di elementi.

I sistemi di numerazione possono essere classificati in base a due criteri principali:

- **Base**: la base di un sistema di numerazione è il numero di simboli utilizzati per rappresentare i numeri. Il sistema di numerazione più comune, quello decimale, ha una base di 10, quindi utilizza 10 simboli: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- **Posizionamento**: un sistema di numerazione è posizionale se il valore di un simbolo dipende dalla sua posizione all'interno del numero. Il sistema decimale è posizionale: ad esempio, il numero 123 significa (1 x 100) + (2 x 10) + (3 x 1), ovvero 123 unità.

I moderni sistemi di numerazione sono posizionali, cioè tutti i simboli (o cifre) vengono ordinati in modo che ognuno abbia peso maggiore rispetto al simbolo (cifra) precedente: il valore del numero rappresentato dipende dalle posizioni relative alle cifre che lo compongono.



Esistono anche sistemi di numerazione non posizionali, come il sistema di numerazione romano, che utilizza simboli diversi per rappresentare valori diversi. Ad esempio, il numero 5 si rappresenta con il simbolo “V”, il numero 10 si rappresenta con il simbolo “X”, il numero 100 si rappresenta con il simbolo "C", il numero 500 con il simbolo "D", e il numero 1000 con il simbolo "M".

Sistema di numerazione binario

Il sistema di numerazione binario si basa su due soli simboli: 0 e 1 ed è quindi un sistema numerico posizionale in base 2.

Quando si usano diversi sistemi di numerazione per evitare ambiguità si scrive la base come pedice del numero, quindi in questo caso per indicare un numero binario si mette 2 come pedice, mentre per indicare un numero decimale si mette 10 come pedice.

Come in tutti i sistemi di numerazione ogni cifra possiede un peso che è determinato dalla sua posizione.

In informatica il sistema binario è utilizzato per la rappresentazione interna dell'informazione dalla quasi totalità degli elaboratori elettronici, in quanto le caratteristiche fisiche dei circuiti digitali rendono molto conveniente la gestione di due soli valori, rappresentati fisicamente da due diversi livelli di tensione elettrica. Tali valori assumono convenzionalmente il significato numerico di 0 e 1 (come la corrente che attraversa o non attraversa un circuito) o quelli di vero e falso della logica booleana.

Di seguito viene riportata, la corrispondente conversione in numero binario per le prime venti cifre del sistema decimale e le potenze del 2 fino alla decima.

Decimale	**Binario**
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100

Potenze del 2	
2^0 =	1
2^1 =	2
2^2 =	4
2^3 =	8
2^4 =	16
2^5 =	32
2^6 =	64
2^7 =	128
2^8 =	256
2^9 =	512
2^{10} =	1.024

Conversione da decimale a binario

Per convertire un numero decimale in binario si divide il numero decimale per 2 scrivendo come risultato solo la parte intera e riportando il resto della divisione, il procedimento viene ripetuto fino a quando si ottiene 0 come risultato finale.

Alla fine si considerano i resti delle varie divisioni leggendoli al contrario.

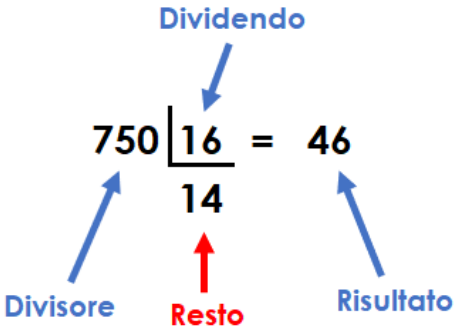
Esempio:

Rappresentazione decimale	13_{10}
Divisioni	13 : 2 = 6 : 2 = 3 : 2 = 1 : 2 = 0
Resti	1 0 1 1

$$13_{10} = 1101_2$$

Calcolo del resto di una divisione

Per capire come calcolare il resto di una divisione, prendere in considerazione il seguente esempio:



La procedura per determinare il resto di una divisione è la seguente:

1. Calcolare il risultato della divisione, scrivendo solamente la parte intera
2. Resto = Divisore - (Dividendo X Risultato)

Applicato all’esempio diventa:

1. 750 : 16 = 46
2. 750 - (16 X 46) = 750 - 736 = 14

Ne consegue che il resto è uguale a 14

Conversione da binario a decimale

Dato che il sistema binario è a “base 2”, ogni cifra di un numero corrisponde a una potenza di 2. Per convertire un numero binario in decimale, si moltiplica, partendo da destra verso sinistra, ogni cifra che lo compone moltiplicato per la potenza di due relativa alla posizione assunta. In particolare la cifra più a destra si trova in posizione 0 e man mano che ci si sposta verso sinistra la sua posizione viene incrementata di 1.
La cifra più a destra prende il nome di cifra meno significativa, mentre la cifra più a sinistra prende il nome di cifra più significativa. Alla fine si sommano i risultati di tutte le moltiplicazioni delle singole cifre per la relativa potenza.

Esempio:

Rappresentazione binaria 111001_2

$$(1 \cdot 2^5) + (1 \cdot 2^4) + (1 \cdot 2^3) + (0 \cdot 2^2) + (0 \cdot 2^1) + (1 \cdot 2^0) =$$
$$(1 \cdot 32) + (1 \cdot 16) + (1 \cdot 8) + (0 \cdot 4) + (0 \cdot 2) + (1 \cdot 1) =$$
$$32 + 16 + 8 + 1$$
$$87$$

$$111001_2 = 57_{10}$$

Rappresentazione in virgola fissa

La rappresentazione di un valore in virgola fissa non si distanzia molto dalla rappresentazione degli interi: si sta sempre lavorando con un sistema di numerazione posizionale e in particolare con quello binario.

Anche in questo caso ogni cifra di un numero assumerà un valore dato dal valore della cifra in sé opportunamente pesato dalla posizione in cui si trova la cifra.

Numeri Negativi in binario

Per rappresentare i numeri negativi in formato binario, si usa un metodo chiamato ‘complemento a 2’.

Dati numeri di N cifre binarie, volendo rappresentare i numeri sia negativi che positivi, si divide il range in 2 parti: negativi, zero, positivi, per cui avremo la meta dei numeri disponibili normalmente.

Con 8 bit, in rappresentzione classica (solo numeri positivi), possiamo rappresentare 255 numeri positivi piú lo zero, quindi da 0 a 255.

Con 8 bit, in rappresentazione complemento a 2 , possiamo rappresentare 128 numeri negativi, lo zero, e 127 numeri positivi, cosi da riempiure il range di 256 numeri.

Quindi il range sara -128, 127.

Nella rappresentazione complemento a 2, il primo bit esprime il segno:

- **0: segno positivo**, rappresentazione identica a quella classica
- **1: segno negativo**, rappresentazione complemento a 2

Ad esempio il numero positivo **10** , in binario **00001010**, ha segno positivo, infatti il **primo bit è 0**.

Invece, il numero **-10**, in binario con complemento a 2 è 1110110, e si ottiene partendo dal valore assoluto, 10, rappresentato in binario:

00001010

a cui poi si invertono (si ‘complementano’) tutti i bit:

11110101

a cui poi si somma 1:

11110101 +

00000001 =

————————

11110110

Come vediamo, il numero -10, negativo, ha il primo bit a 1, il che esprime il segno negativo.

Byte

Il byte è l'unità di misura delle capacità di memoria, o in altri termini la misura della quantità di informazioni.

Per capire cos'è il byte si deve fare riferimento alla logica che regola il funzionamento dei computer. I calcolatori operano usando la base 2 (sistema binario), la quale utilizza solamente due cifre: 0 e 1. Ciascuna di tali cifre prende il nome di bit.

Il byte è composto da 8 bit ed è pertanto in grado di assumere $2^8 = 256$ possibili valori (da 0 a 255). Per definizione 1 byte, indicato con la lettera maiuscola B, ed è un'unità di misura della quantità di informazioni e corrisponde a una sequenza formata da 8 bit consecutivi.

I multipli del byte sono così indicati:

NOME	SIMBOLO	MULTIPLO
yottabyte	YB	2^{80}
zettabyte	ZB	2^{70}
exabyte	EB	2^{60}
petabyte	PB	2^{50}
terabyte	TB	2^{40}
gigabyte	GB	2^{30}
megabyte	MB	2^{20}
chilobyte	kB	2^{10}
byte	**B**	2^0
bit	*bit*	*1/8*

Si noti che il byte, essendo formato da 8 bit e quindi da grandezze binarie, per salire o scendere di un gradino si usa la potenza del 2 e non quella del 10 come può essere per il metro o per il chilo.

Ne consegue che:

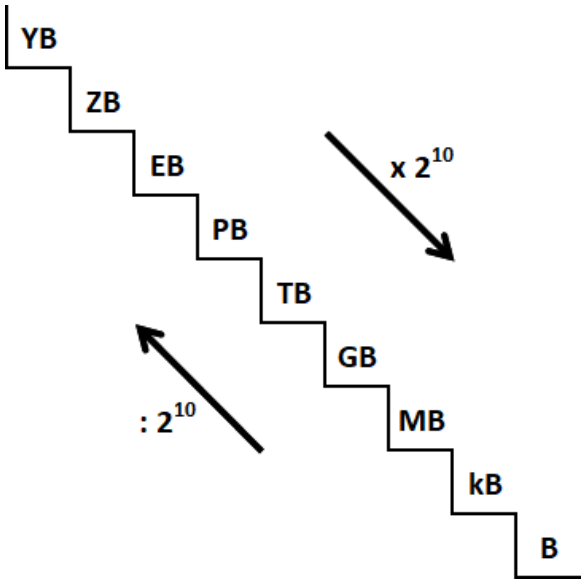
1 kB = 2^{10} B = 1.024 B

1 MB = 2^{20} B = 1 x 1.024 x 1.024 = 1.048.576 B = 1.024 kB

1 GB = 2^{30} B = 1 x 1.024 x 1.024 x 1.024 = 1.073.741.824 B = 1.048.576 kB = 1.024 MB

1 TB = 2^{40} B = 1 x 1.024 x 1.024 x 1.024 x 1.024 = 1.099.511.627.776 B = 1.073.741.824 kB = 1.048.576 MB = 1.024 GB

Per effettuare una conversione tra multipli del byte, disegnare una scala con una serie di gradini, su cui vanno riportati ordinatamente i simboli dei multipli binari del byte.



Per svolgere una qualsiasi conversione si deve contare il numero di gradini che separano le due unità di misure coinvolte, per poi:

- moltiplicare per 2 elevato al numero di gradini per 10 se si scende;
- dividere per 2 elevato al numero di gradini per 10 se si sale.

Esempio:

Convertire 0,5 TB in MB

- Si deve scendere di 2 gradini, pertanto:
- $0,5 \times 2^{2 \times 10} = 0,5 \times 2^{20} = 0,5 \times 1024 \times 1024 = 524.288$ MB

Convertire 3.298.534.883.328 kB in TB

- Si deve salire di 3 gradini, pertanto:
- $3.298.534.883.328 : 2^{3 \times 10} = 3.298.534.883.328 : 2^{30} = 3.298.534.883.328 : 1024 : 1024 = 3.072$ TB

Bit

Il bit è una cifra binaria e può contenere i due simboli del sistema binario: zero (0) e uno (1) ed è considerata la più piccola unità di informazione. Fin dagli albori dell'informatica il sistema binario è il linguaggio madre degli elaboratori elettronici e dei computer. Le ragioni di questa scelta sono molto semplici. Lo stato "zero" e "uno" possono essere realizzati con la presenza o l'assenza di tensione elettrica in un circuito.

Per indicare un bit si usa la lettera minuscola “b” o più semplicemente la parola “bit”.

Come detto in precedenza 8 bit equivalgono a 1 Byte.

I multipli del bit seguono la stessa logica del byte, e sono così indicati:

NOME	SIMBOLO	MULTIPLO
yottabit	Ybit	2^{80}
zettabit	Zbit	2^{70}
exabit	Ebit	2^{60}
petabit	Pbit	2^{50}
terabit	Tbit	2^{40}
gigabit	Gbit	2^{30}
megabit	Mbit	2^{20}
chilobit	kbit	2^{10}
bit	**bit**	2^{0}

Si noti che il bit, essendo una cifra binaria, per salire o scendere di un gradino si usa la potenza del 2 (come per il byte) e non quella del 10 come può essere per il metro o per il chilo.

Ne consegue che:

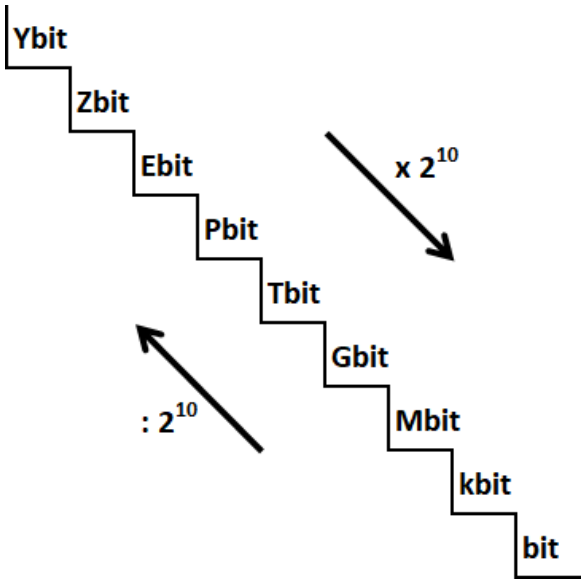
1 kbit = 2^{10} bit = 1.024 bit

1 Mbit = 2^{20} bit = 1 x 1.024 x 1.024 = 1.048.576 bit = 1.024 kbit

1 Gbit = 2^{30} bit = 1 x 1.024 x 1.024 x 1.024 = 1.073.741.824 bit = 1.048.576 kbit = 1.024 Mbit

1 Tbit = 2^{40} bit = 1 x 1.024 x 1.024 x 1.024 x 1.024 = 1.099.511.627.776 bit = 1.073.741.824 kbit = 1.048.576 Mbit = 1.024 Gbit

Per effettuare una conversione tra multipli del byte, disegnare una scala con una serie di gradini, su cui vanno riportati ordinatamente i simboli dei multipli binari del byte.



Per svolgere una qualsiasi conversione si deve contare il numero di gradini che separano le due unità di misure coinvolte, per poi:

- moltiplicare per 2 elevato al numero di gradini per 10 se si scende;
- dividere per 2 elevato al numero di gradini per 10 se si sale.

Esempio:

Convertire 0,5 Tbit in Mbit

- Si deve scendere di 2 gradini, pertanto:
- $0,5 \times 2^{2x10} = 0,5 \times 2^{20} = 0,5 \times 1024 \times 1024 = 524.288$ Mbit

Convertire 3.298.534.883.328 kbit in Tbit

- Si deve salire di 3 gradini, pertanto:
- $3.298.534.883.328 : 2^{3x10} = 3.298.534.883.328 : 2^{30} = 3.298.534.883.328 : 1024 : 1024 = 3.072$ Tbit

Rappresentazione informazioni binarie

Ai fini dei linguaggi di programmazione è comune raggruppare sequenze di bit in entità più vaste che possono assumere valori in intervalli assai più ampi di quello consentito da un singolo bit. Questi raggruppamenti contengono generalmente un numero di stringhe binarie pari a una potenza binaria, pari cioè a 2^n; il più noto è il byte corrispondente a 8 bit, che costituisce l'unità di misura più utilizzata in campo informatico. In altri ambiti diversi dalla programmazione questi raggruppamenti sono poco usati.

Altri raggruppamenti di questo tipo sono i seguenti:

NUMERO DI BIT	TERMINE DI RAGGRUPPAMENTO
1	bit
4	nibble
8	Byte
16	word
32	double word
64	quad word

Hertz

L'hertz (simbolo Hz) è l'unità di misura del Sistema Internazionale della frequenza.

La frequenza rappresenta il numero di cicli o oscillazioni completate in un secondo. Quindi, un Hertz equivale a un ciclo al secondo. Ad esempio, se qualcosa ha una frequenza di 10 Hertz, significa che compie 10 cicli o oscillazioni in un secondo.

In informatica, gli hertz hanno molteplici utilizzi, ad esempio vengono utilizzati per misurare la frequenza di clock di un processore o il refresh rate di uno schermo. Ad esempio:

- **La frequenza della CPU**, rappresenta quante operazioni elementari (o istruzioni) un processore riesce ad eseguire nell’arco di un secondo.
  Una CPU con una frequenza più alta può eseguire più istruzioni al secondo, il che significa che può elaborare le informazioni più velocemente.
  Una CPU con una frequenza di clock di 3 GHz può eseguire 3 miliardi di operazioni elementari al secondo.
- **Il refresh rate di un monitor** è il numero di volte al secondo in cui l'immagine viene ridisegnata (o aggiornata) sullo schermo. Un refresh rate più alto produce un'immagine più fluida.
  Uno schermo con un refresh rate di 144 Hz ridisegna l’immagine 144 volte nell’arco di un secondo.

Utilizzo del sistema binario

Il sistema binario è utilizzato per la rappresentazione interna dell'informazione dalla quasi totalità degli elaboratori elettronici, in quanto le caratteristiche fisiche dei circuiti digitali rendono molto conveniente la gestione di due soli valori, rappresentati fisicamente da due diversi livelli di tensione elettrica. Oltre all’uso interno che ne fa un calcolatore, è di particolare interesse lo studio del codice ASCII, che permette di capire la quantità di informazioni che possono essere rappresentate con 1 byte (ossia 8 bit che equivalgono a 256 combinazioni).

Codice ASCII

Per permettere la comunicazione tra uomo e calcolatore senza dover ricorrere al codice binario, è stato introdotto il sistema di codifica ASCII. Tale sistema associa a ciascuno dei possibili 256 valori assunti dal byte una determinata lettera, numero o carattere speciale. Ad esempio la lettera A maiuscola corrisponde al byte 01000001.

La sigla “ASCII” sta per American Standard Code for Information Interchange, cioè “Standard americano per lo scambio di informazioni”.

Per effettuare questa codifica (o conversione) si una la tabella ASCII che non fa altro che associare a ciascun carattere un codice numerico.

La tabella è formata da 256 numeri decimali che vanno da 0 a 255 (da 00000000 a 11111111 in binario).

- I numeri da 0 a 31 sono riservati per dei segnali di controllo.
- I numeri da 32 a 127 costituiscono il set di caratteri standard, ossia tutte le lettere dell’alfabeto latino in minuscolo e maiuscolo, i numeri e i simboli maggiormente usati.

- I numeri che vanno da 128 a 255 costituiscono il set di caratteri estesi che comprendono caratteri speciali, matematici, grafici e di lingue straniere (come ad esempio æ e Ç).

La tabella ASCII è organizzata in tre colonne:

- la prima colonna contiene il corrispondente numero binario associato ad un determinato carattere;
- la seconda colonna contiene il corrispondente numero decimale associato ad un determinato carattere;
- la terza colonna contiene il carattere corrispondente.

Ad esempio, la lettera "A" ha il codice ASCII 65, il numero "1" ha il codice ASCII 49 e il simbolo "@" ha il codice ASCII 64.

Tabella ASCII Standard

Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
00000000	0	Null	00100000	32	Spc	01000000	64	@	01100000	96	`
00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	Start of text	00100010	34	”	01000010	66	B	01100010	98	b
00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	End of transmit	00100100	36	$	01000100	68	D	01100100	100	d
00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	Audible bell	00100111	39	’	01000111	71	G	01100111	103	g
00001000	8	Backspace	00101000	40	(	01001000	72	H	01101000	104	h
00001001	9	Horizontal tab	00101001	41	)	01001001	73	I	01101001	105	i
00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	Neg. acknowledge	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	Escape	00111011	59	;	01011011	91	[	01111011	123	{
00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	|
00011101	29	Group separator	00111101	61	=	01011101	93	]	01111101	125	}
00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	Unit separator	00111111	63	?	01011111	95	_	01111111	127	Del

Le combinazioni da 0 a 31 possono risultare differenti a seconda del software utilizzato

Se si sta usando Windows si può ottenere ogni carattere ASCII tenendo premuto il tasto Alt e digitando il codice decimale corrispondente col tastierino numerico (se il tastierino numerico non fosse attivo, premere prima il tasto “Num lock” o “Bloc Num” per attivarlo).
per esempio la chiocciola @ si ottiene digitando 64 mentre si tiene premuto il tasto Alt.

È utile poiché, ad esempio nella tastiera con layout italiano, mancano alcuni simboli, come l'apice (96), le parentesi graffe (123,125) o la tilde (126) e lettere accentante in maiuscolo (À, È, Ì, Ò, Ù).

Tabella ASCII Estesa

Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
10000000	128	Ç	10100000	160	á	11000000	192	+	11100000	224	Ó
10000001	129	ü	10100001	161	í	11000001	193	-	11100001	225	ß
10000010	130	é	10100010	162	ó	11000010	194	-	11100010	226	Ô
10000011	131	â	10100011	163	ú	11000011	195	+	11100011	227	Ò
10000100	132	ä	10100100	164	ñ	11000100	196	-	11100100	228	õ
10000101	133	à	10100101	165	Ñ	11000101	197	+	11100101	229	Õ
10000110	134	å	10100110	166	ª	11000110	198	ã	11100110	230	µ
10000111	135	ç	10100111	167	º	11000111	199	Ã	11100111	231	þ
10001000	136	ê	10101000	168	¿	11001000	200	+	11101000	232	Þ
10001001	137	ë	10101001	169	®	11001001	201	+	11101001	233	Ú
10001010	138	è	10101010	170	¬	11001010	202	-	11101010	234	Û
10001011	139	ï	10101011	171	½	11001011	203	-	11101011	235	Ù
10001100	140	î	10101100	172	¼	11001100	204	¦	11101100	236	ý
10001101	141	ì	10101101	173	¡	11001101	205	-	11101101	237	Ý
10001110	142	Ä	10101110	174	«	11001110	206	+	11101110	238	¯
10001111	143	Å	10101111	175	»	11001111	207	¤	11101111	239	´
10010000	144	É	10110000	176	_	11010000	208	ð	11110000	240	-
10010001	145	æ	10110001	177	_	11010001	209	Ð	11110001	241	±
10010010	146	Æ	10110010	178	_	11010010	210	Ê	11110010	242	‗
10010011	147	ô	10110011	179	¦	11010011	211	Ë	11110011	243	¾
10010100	148	ö	10110100	180	¦	11010100	212	È	11110100	244	¶
10010101	149	ò	10110101	181	Á	11010101	213	ı	11110101	245	§
10010110	150	û	10110110	182	Â	11010110	214	Í	11110110	246	÷
10010111	151	ù	10110111	183	À	11010111	215	Î	11110111	247	¸
10011000	152	ÿ	10111000	184	©	11011000	216	Ï	11111000	248	º
10011001	153	Ö	10111001	185	¦	11011001	217	+	11111001	249	¨
10011010	154	Ü	10111010	186	¦	11011010	218	+	11111010	250	·
10011011	155	ø	10111011	187	+	11011011	219	_	11111011	251	¹
10011100	156	£	10111100	188	+	11011100	220	_	11111100	252	³
10011101	157	Ø	10111101	189	¢	11011101	221	¦	11111101	253	²
10011110	158	×	10111110	190	¥	11011110	222	Ì	11111110	254	■
10011111	159	ƒ	10111111	191	+	11011111	223	_	11111111	255	

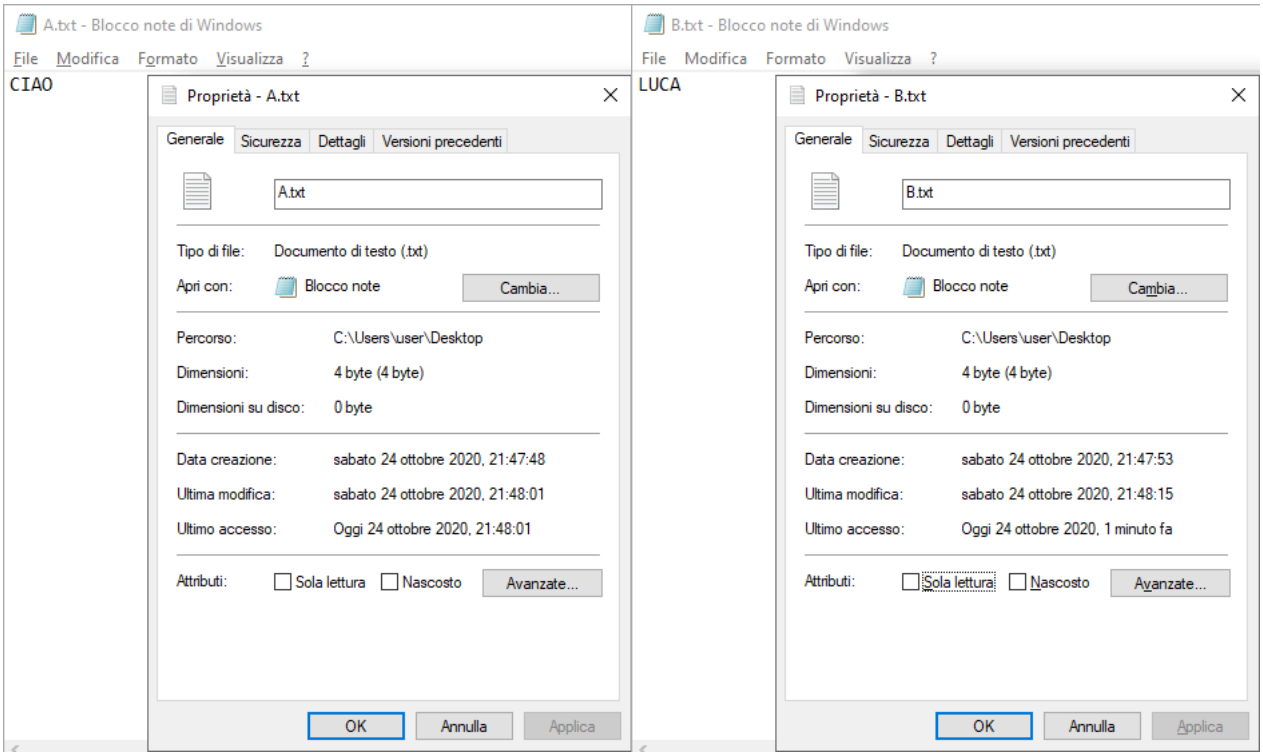
Espansioni del codice ASCII

La prima versione del codice ASCII era a 7 bit, ossia prevedeva 128 combinazioni (riportate nella prima parte della tabella.
Il codice ASCII è stato esteso per supportare caratteri aggiuntivi per la codifica di lingue internazionali. Le estensioni più comuni sono:

- **ASCII esteso (Extended ASCII)**: aggiunge 128 caratteri aggiuntivi, per un totale di 256 caratteri.
- **Unicode:** aggiunge 1.114.112 caratteri, per un totale di 1.114.112 caratteri.

Dimensione di un file di testo con codifica ASCII

Se ad esempio si creano due file con un editor di testo (con un programma tipo blocco note di Windows) e all’interno si scrive rispettivamente le parole “CIAO” e “LUCA”, entrambi i file occuperebbero 4 byte, in quanto ogni carattere occupa esattamente 1 byte ed entrambe le parole scritte all’interno del file sono composte da 4 caratteri. Questo per il discorso appena fatto che 1 byte è formato da 8 bit e quindi da 256 combinazioni che rappresentano dei simboli/caratteri.



L’informazione sulla composizione dei caratteri è racchiusa nella sequenza di bit di ogni singolo byte, infatti si può fare riferimento per ogni carattere al corrispettivo numero in binario racchiuso nel codice ASCII.

Prendendo l’esempio appena fatto, CIAO corrisponde a:

67	73	65	80
ossia			
01000011	01001001	01000001	01001111

Mentre LUCA corrisponde a:

76	85	67	65

ossia
01001100 01010101 01000011 01000001

Utilizzo del sistema esadecimale

Il sistema esadecimale è molto usato in informatica, per la sua relazione diretta tra una cifra esadecimale e quattro cifre binarie.
È usato in differenti linguaggi di programmazione, soprattutto per identificare gli indirizzamenti di memoria, un particolare uso avviene nei programmi di grafica per identificare con poche cifre esadecimali un preciso colore.

Codice HTML

Come detto in precedenza, i codici dei colori sono triplette esadecimali (e quindi una cifra composta da 6 cifre esadecimali), che rappresentano i colori rosso, verde e blu (i tre colori primari).

La base esadecimale è usata nella maggior parte dei programmi di grafica e dai vari browser per navigare su internet, per identificare inequivocabilmente un colore ben specifico.

Il numero esadecimale con valore più alto è FF che equivale al valore decimale 255, mentre il numero esadecimale più basso equivale a 00.
I colori sono indicati con un codice a 6 caratteri preceduto dal simbolo # che indica appunto un colore.
Questo codice composto da 6 cifre esadecimale è anche chiamato “Codice esadecimale dei colori”.

Prendiamo in esame il colore bianco, in base esadecimale è indicato come #FFFFFF è composto da:

- FF ossia Rosso = 255
- FF ossia Verde = 255
- FF ossia Blu = 255

255 parti di rosso, 255 parti di verde e 255 parti blu

Di seguito alcuni esempi di colori con il codice esadecimale corrispondente.

NOME	CODICE	COLORE
Bianco	#FFFFFF	
Nero	#000000	
Rosso	#FF0000	
Blu	#0000FF	
Verde	#009900	
Giallo	#FFFF00	
Magenta	#FF00FF	
Viola	#5C2E91	
Azzurro	#0080FF	

2. “Hardware”

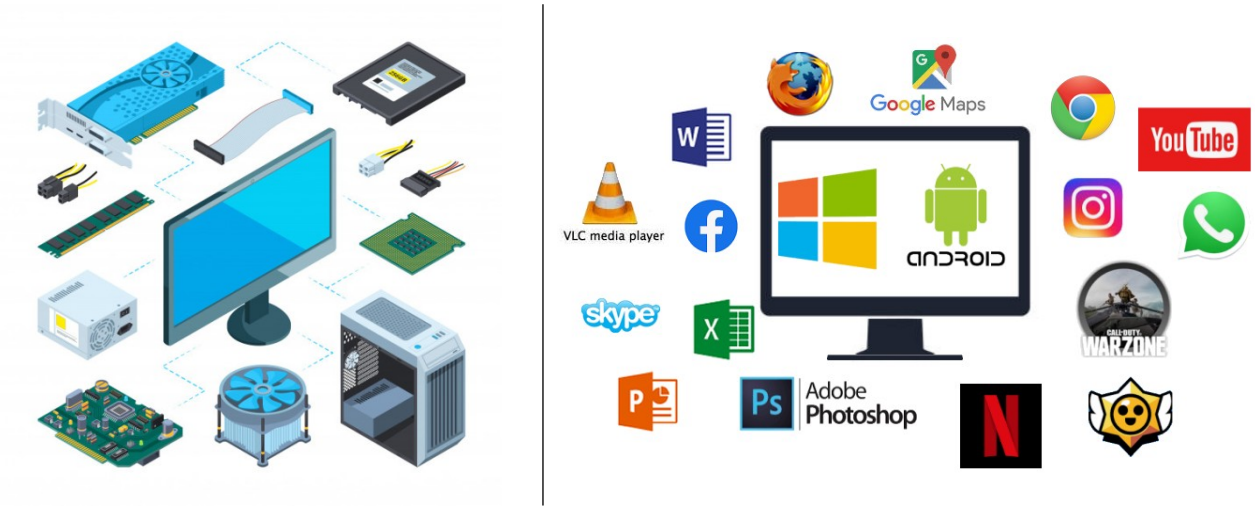
Hardware di un computer

Il computer è un insieme di dispositivi fisici collegati tra loro che hanno lo scopo di ricevere dati dall'esterno, elaborarli seguendo determinate istruzioni contenute in programmi e produrre risultati alla fine del processo di elaborazione.
Come tutte le macchine, non ha nessuna capacità decisionale o discrezionale, ma si limita a compiere determinate azioni secondo procedure prestabilite (programmi).

Volendo semplificare al massimo, possiamo definire un computer come composto dalle seguenti componenti:

1. Componenti Hardware (le parti fisiche e tangibili del computer)
2. Componenti Software (le parti logiche del computer come programmi e applicazioni)

Ne consegue che, in un computer, i componenti hardware e software sono complementari tra loro e indispensabili affinché esso possa funzionare.



Sulla sinistra un esempio di componenti hardware, sulla destra esempio di alcuni software.

Per hardware si intende tutta la parte fisica di un computer, ovvero tutte quelle parti elettroniche, elettriche, meccaniche, magnetiche, ottiche che ne consentono il funzionamento.

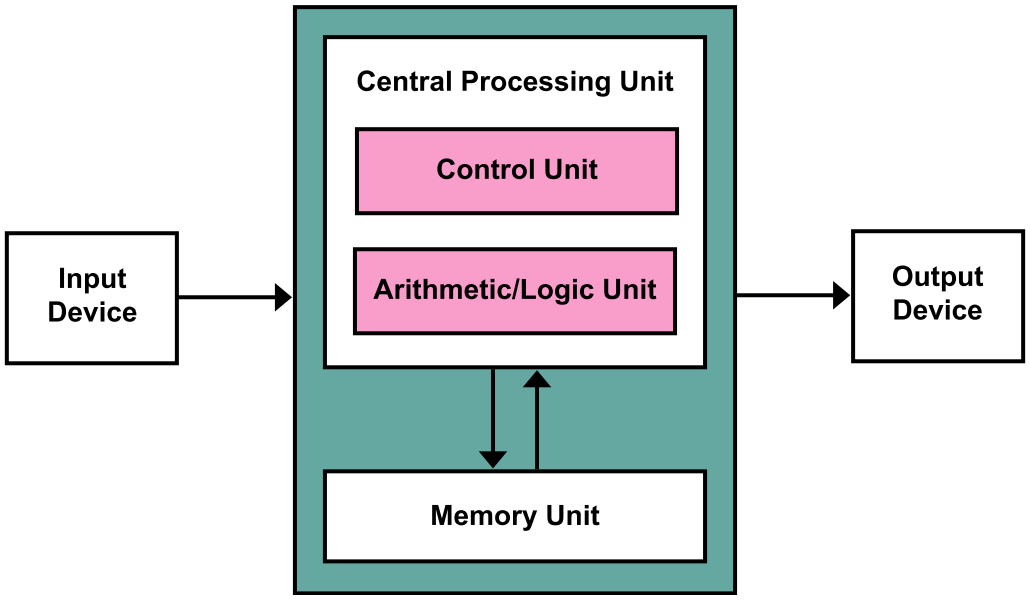
Le varie componenti hardware di un computer sono formate da circuiti elettronici, in grado di elaborare i due stati di attraversamento dell’elettricità corrispondenti ai simboli 0 e 1.

Per essere considerato tale, un computer deve disporre almeno di:

- CPU;
- Memoria;
- Periferiche.

Architettura di von Neumann

Tutti i moderni computer si basano sull’architettura del modello di von Neumann:



Lo schema si basa su cinque componenti fondamentali:

- **CPU** (o processore);
- **Memoria** (dove vengono memorizzati i dati in maniera temporanea o permanente);
- **Unità di input**, tramite la quale i dati vengono inseriti dall’utente nel calcolatore per essere elaborati;
- **Unità di output**, necessaria affinché i dati elaborati possano essere restituiti all'utente;
- **Bus**, un canale che collega tutti i componenti fra loro.

Il suo modello, concepito nel 1945, costituisce ancora oggi il fondamento per la quasi totalità dei computer moderni.

CPU

La CPU (Central Processing Unit, o unità centrale di elaborazione) ha come compito fondamentale l’esecuzione delle istruzioni dei programmi che il computer esegue.

Si avvale di una serie di componenti, al suo interno, che svolgono dei compiti specifici:

- ALU: Unità Aritmetico Logica
- CU: Unità di Controllo, coordina l’acquisizione e l’esecuzione delle istruzioni
- Clock: un orologio di sistema che sincronizza componenti e istruzioni.
- Registri interni: memorizzano temporaneamente istruzioni e risultati parziali delle operazioni.

Per poter essere eseguito, un programma deve essere caricato nella memoria RAM dalla memoria di massa dove è memorizzato in modo permanente.

Una volta che il programma risiede nella memoria RAM, viene eseguita ciclicamente un'istruzione per volta, la singola istruzione. per poter essere eseguita, deve essere prelevata (**Fetch**) dalla memora RAM alla CPU, dove viene codificata (**Decode**) e quindi eseguita (**Execute**).

Il ciclo **Fetch-Decode-Execute** viene chiamato ciclo del processore (o clock) e viene eseguito in modo ciclico dalla CU, mentre la ALU esegue operazioni di calcolo quando necessario.

Tra le caratteristiche di una CPU bisogna ricordare:

- Frequenza di Clock: è la velocità della CPU, si misura in multipli dell’Hertz (Hz), attualmente tra i 2 e i 5 GHz
- Numero di core: single, dual, quad, octa
- Numero di thread
- Presenza e tipo di cache: L1 (dentro la CPU, veloce), L2 (dentro o fuori CPU), L3 (tra CPU e RAM)
- Architettura (o set di istruzioni): x86, x86-64, ARM, PowerPC, MIPS, SPARC, ecc..

Frequenza di Clock

La frequenza di clock (o velocità di clock) rappresenta il numero di operazioni elementari elaborate nell’arco di un secondo ed è misurata in GHz (gigahertz).

In generale, una frequenza di clock superiore significa che la CPU è più veloce. La CPU elabora ogni secondo molte istruzioni (calcoli di basso livello, come l'aritmetica) provenienti da programmi diversi.

Ad esempio, una CPU con una velocità di clock di 3,2 GHz esegue 3,2 miliardi di cicli al secondo (o più semplicemente 3,2 operazioni elementari in un secondo). In passato la velocità delle CPU era misurata in megahertz e quindi in milioni di cicli al secondo.

Numero di core

Una CPU multi-core è un insieme di unità processanti autonome in un unico chip, che condividono le varie risorse del sistema (cache, bus, e memoria centrale).

In linea teorica, disporre, ad esempio, di due core significa avere una potenza processuale duplicata, ma solo nel caso in cui le applicazioni siano ottimizzate per essere elaborate da una CPU multi-core.

In altre parole, una CPU multi-core ha la possibilità di eseguire in parallelo contemporaneamente più istruzioni, in linea teorica una CPU dual core a 2 GHz ha la capacità di eseguire 4 miliardi di operazioni elementari al secondo

Inoltre, una CPU multi-core non necessita di più energia rispetto a un single core, ne consegue quindi oltre una maggiore potenza di calcolo anche una maggiore autonomia per i dispositivi mobili.

Numero di thread

Una CPU multi-thread rappresenta la capacità da parte di ogni singolo core di elaborare simultaneamente più operazioni elementari.

Un thread, per semplificare moltissimo, è una parte dalle istruzioni che una CPU deve elaborare per eseguire un compito (o, meglio, un "processo"). Una CPU multi-thread quindi può suddividere un processo in più thread ed elaborarli parallelamente per offrire più velocemente il risultato atteso.

Anche se la CPU è una sola, il sistema operativo vede due "unità logiche" perché essa può effettivamente elaborare due thread contemporaneamente. Nelle CPU multi-core, il numero di thread si riferisce per ogni core del processore. Ad esempio, una CPU quad core/dual thread è capace di processare 8 operazioni simultaneamente in quanto ha 4 core fisici (e quindi 4 operazioni in simultanea) e ogni singolo core viene visto come 2 core separati, ne consegue che 4 core x 2 thread = 8 operazioni simultanee.

Cache della CPU

La cache utilizzata dalla CPU di un computer ha il compito di ridurre il tempo medio d'accesso alla memoria; è un tipo di memoria di dimensioni ridotte, ma molto veloce, che mantiene copie dei dati ai quali si fa più frequentemente accesso nella più capiente memoria principale.

Su una CPU possono essere presenti 3 livelli di cache e sono organizzati in modo gerarchico:

La cache L1 (cache di primo livello): è interna al processore ed è spesso differenziata in cache per dati e cache per istruzioni. La cache L1 tende ad essere dell'ordine di 32-64 kB.

La cache L2 (cache di secondo livello): è generalmente più grande ma leggermente più lenta della cache L1 ed è legata a un core della CPU. I processori recenti tendono ad avere fino a 512 kB di cache per core e questa cache non fa distinzione tra istruzioni e cache di dati in quanto è una cache unificata.

La cache L3 (cache di terzo livello): è condivisa da tutti i core presenti sulla CPU ed è molto più grande e più lenta rispetto alla cache L2, ma è ancora molto più veloce rispetto alla memoria principale. Può essere sia interna che esterna ma può anche non essere presente (soprattutto nelle CPU meno recenti). Non viene differenziata fra dati e istruzioni. La cache L3 tende ad essere dell'ordine di 16-32 MB.

Architettura (Instruction set architecture)

L’architettura di una CPU (in inglese, instruction set architecture, di cui l’acronimo ISA) definisce l'insieme di istruzioni base (o operazioni elementari) che la CPU può compiere e che costituiscono dunque il suo linguaggio macchina, a partire dal quale vengono scritti i relativi programmi nei vari linguaggi di programmazione.

L’architettura di una CPU può essere classificata in diversi modi. Una classificazione comune è per complessità architettonica. Un computer con set di istruzioni complesso (CISC) ha molte istruzioni specializzate, alcune delle quali possono essere utilizzate solo raramente in programmi pratici. Un computer con un set di istruzioni ridotto (RISC) semplifica il processore implementando in modo efficiente solo le istruzioni utilizzate frequentemente nei programmi, mentre le operazioni meno comuni sono implementate come combinazione di più operazioni elementari, con il conseguente tempo di esecuzione del processore aggiuntivo compensato da un uso poco frequente e con il

vantaggio di abbattere notevolmente i consumi energetici (fattore molto utile per dispositivi mobili come smartphone e tablet)

Attualmente tutti i computer (ad eccezione dei Mac prodotti dal 2020 in poi) montano dei processori con un set di istruzioni di tipo “**x86**” (famiglie di prodotti realizzati da Intel e AMD, contenenti istruzioni di tipo CISC), mentre tutti gli smartphone e i Mac prodotti dal 2020 in poi utilizzano processori con un set di istruzioni di tipo “**ARM**” (famiglie di prodotti realizzati da produttori di terze parti sotto licenza da parte di ARM Holdings, contenenti istruzioni di tipo RISC).
Oltre queste due architetture ne esistono altre, ma ormai sono in disuso in ambito consumer, come ad esempio l’architettura PowerPC, usata per i Mac fino al 2006 o dalla Playstation 3 o Xbox 360.

Memoria

Le memorie sono destinate al salvataggio di dati e alla lettura di essi. Nell'architettura dei calcolatori, si distinguono due tipi di memoria:

- **memoria principale (o memoria centrale)**, destinata a lavorare a più stretto contatto con la CPU, costituita fondamentalmente da memoria RAM, memoria ROM, memoria Cache;
- **memoria secondaria (o memoria di massa)**, destinata a memorizzare i dati in maniera permanente, come ad esempio gli hard disk, SSD, ma anche supporti rimovibili come CD, DVD, memorie flash di ogni tipo, penne USB ed altro ancora.

Sono caratterizzate dai seguenti parametri:

- **Capacità**: quantità di spazio disponibile
- **Volatilità**: indica se la memoria può o meno mantenere il dato in assenza di corrente
- **Tempo di accesso**: intervallo di tempo necessario per completare una lettura o una scrittura
- **Velocità di trasferimento**: quantità di dati trasferiti nell’unità di tempo
- **Costo per bit:** prezzo

La quantità di spazio di memoria è misurata in byte e multipli.

Memorie principali

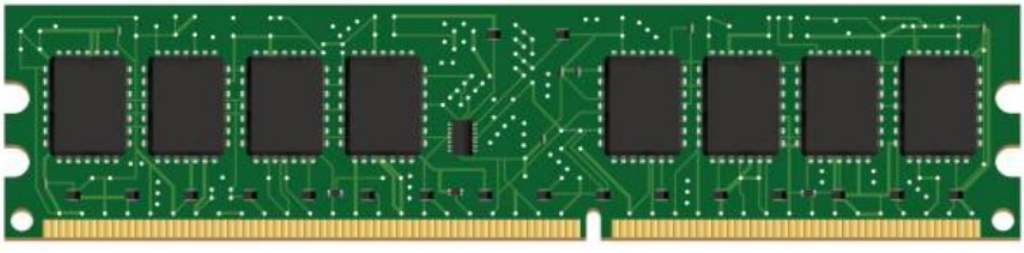
RAM (Random Access Memory)

La RAM (Random Access Memory) memorizza i dati in maniera temporanea in attesa di essere elaborati dalla CPU. Di fatto contiene, durante l’esecuzione dei programmi, le istruzioni dei programmi che, via via, la CPU deve eseguire.

Essa è una **memoria volatile**, ossia i dati vengono persi allo spegnimento o al riavvio del sistema, e viene chiamata ad **accesso casuale,** poiché il tempo di accesso è indipendente dalla posizione fisica del dato.

Il limite massimo di RAM installabile è dato dall’architettura del processore e dal sistema operativo utilizzato:

- 32 bit = 2^{32} indirizzi = 4 GB
- 64 bit = 2^{64} indirizzi = 16 EB (Exabyte ossia 16.000.000 GB)



ROM (Read Only Memory)

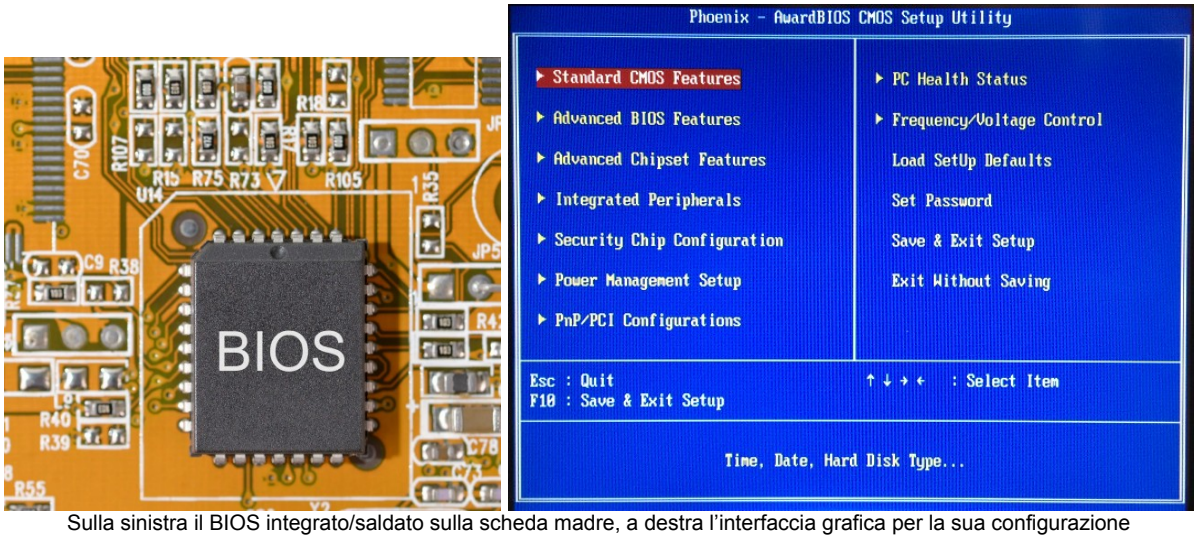
La ROM è una memoria in sola lettura, non è pertanto possibile memorizzare dati in essa.

La ROM contiene tutti i parametri di configurazione, al fine di far interagire tutti i componenti hardware e software tra di loro.

All’interno della ROM sono memorizzati dei particolari programmi:

- **BIOS (Basic I/O System)**, presente su tutti i computer prodotti fino al 2010, che viene eseguito dalla CPU all'accensione del computer, quando la RAM è vuota;
- **UEFI (Unified Extensible Firmware Interface)**, che è in pratica una evoluzione del BIOS, in quanto oltre a far correttamente avviare il dispositivo e far dialogare l’hardware con il software, integra altre funzioni avanzate;
- **Firmware**, presente su tutti i dispositivi mobili (come smartphone e tablet) e su tutti i sistemi embedded (come elettrodomestici in generale, ad esempio lavatrici, lavastoviglie, forni a microonde, televisioni, decoder, ecc…)

Lo scopo del BIOS (come di qualsiasi altro firmware) è quello di fare interagire i vari componenti hardware tramite l'implementazione di protocolli di comunicazione o interfacce di programmazione. Rappresenta di fatto il punto di incontro fra le componenti logiche e fisiche di un dispositivo elettronico, ossia tra software e hardware.

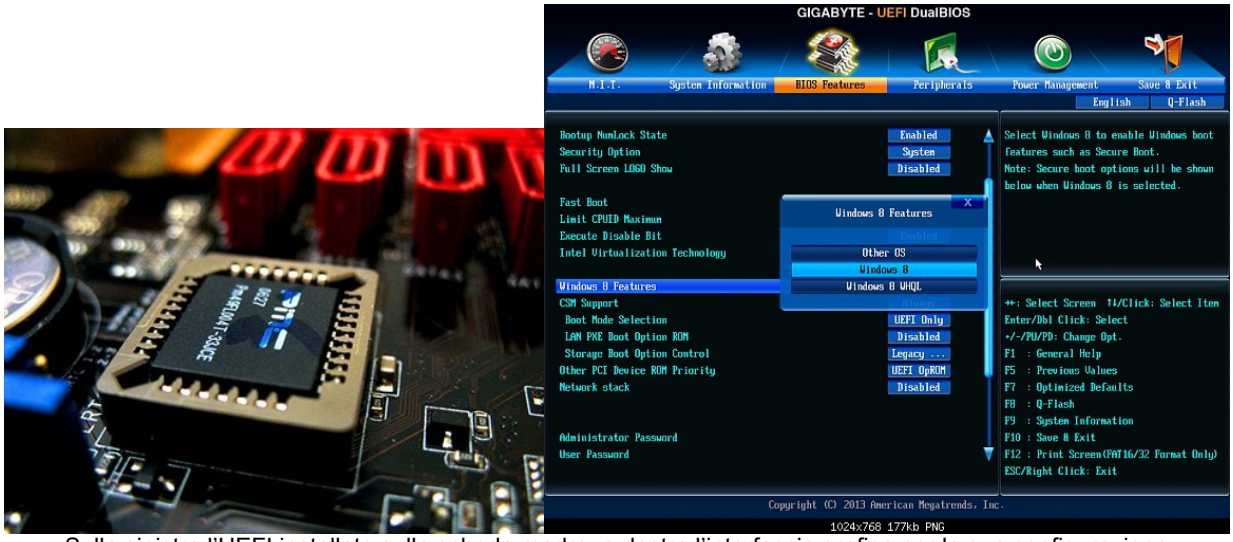


Sulla sinistra il BIOS integrato/saldato sulla scheda madre, a destra l’interfaccia grafica per la sua configurazione

Nei personal computer, il BIOS a partire dal 2010 è stato progressivamente sostituito da UEFI (Unified Extensible Firmware Interface), che è in pratica una sua evoluzione, in quanto oltre a far correttamente avviare il dispositivo e far dialogare l’hardware con il software, integra altre funzioni:

- **Gestione della licenza sistema operativo:** funzione di registrazione della licenza digitale del sistema operativo in un'area di memorizzazione specifica del chip, in modo da non dover riattivare il sistema operativo a seguito, ad esempio, di una formattazione.
- **Gestore di avvio:** o boot manager, che prima era memorizzato solo sulla memoria di massa.
- **Avvio protetto:** o secure boot, che consente di bloccare e impedire che virus o altre minacce possano essere avviati all'accensione/riavvio della macchina costituendo così un grave rischio per il sistema.
- **Garanzia produttore:** La piattaforma permette al produttore della macchina di memorizzare i termini dell'eventuale garanzia fornita, in primis la data di scadenza. In questo modo l'utente

può monitorare lo stato della garanzia sia attraverso il menù del firmware che attraverso strumenti del costruttore (applicazione o sito web).



Sulla sinistra l’UEFI installato sulla scheda madre, a destra l’interfaccia grafica per la sua configurazione

Memorie secondarie

Hard Disk

Un disco rigido o disco fisso, (in inglese hard disk drive, abbreviato comunemente in hard disk e con le sigle HDD, HD) indica un dispositivo di memoria di massa di tipo magnetico che utilizza uno o più dischi magnetizzati per l'archiviazione di dati e applicazioni (file, programmi e sistemi operativi).

Come tutte le memorie di massa ha come scopo primario la memorizzazione permanente delle informazioni al suo interno. Ha una capacità più elevata rispetto alla RAM ma è notevolmente più lenti in termini di velocità di trasferimento.

Caratteristiche principali:

- Velocità: dei "piatti" dell’hard disk, 5400, 7200, 10000, 15000 rpm (revolutions per minute, giri al minuto);
- Dimensione: 2,5" (portatili) o 3,5" (desktop);
- Capacità: 500GB, 1TB, 2TB, 4TB, ecc;
- Interfaccia di collegamento: SATA se interno, USB o Thunderbolt se esterno

Caratteristiche secondarie:

- Tempo di accesso: tempo medio necessario per reperire un dato (es: con HD a 7200 rpm, circa 9 ms);
- Velocità di trasferimento: espressa in MB/s, indica la quantità di dati fornita dall’HD in un secondo;
- Rumorosità emessa, espressa in dB;

SSD (Solid State Drive)

Un disco a stato solido (in acronimo SSD dal corrispondente termine inglese solid-state drive) è un dispositivo di memoria di massa basato su semiconduttore, che utilizza memoria allo stato solido (solid-state storage) come le memorie flash, per l'archiviazione dei dati.

Svolge la medesima funzione del tradizionale hard disk, con alcuni vantaggi:

- Rumorosità assente, non essendo presente alcun componente meccanico (motore e disco magnetico) di rotazione, al contrario degli HD tradizionali;
- Minore possibilità di guasto;
- Minori consumi elettrici durante le operazioni di lettura e scrittura;
- Tempi di accesso e archiviazione ridotti: si lavora nell'ordine dei decimi di millisecondo (SSD: 0.1 ms, HDD: 5-10 ms);
- Maggior velocità di trasferimento dati (600 MB/s con SATA 3, 3 GB/s con PCIe 3.0);
- Maggiore resistenza agli urti, in quanto non ci sono parti meccaniche al suo interno;
- Minore produzione di calore;
- Interfaccia di collegamento: SATA o PCI-Express (PCIe), in particolare gli SSD SATA hanno la stessa identica forma, dimensione ed interfaccia di collegamento dei dischi rigidi SATA da 2,5" e sono pertanto interscambiabili con essi senza installare componenti hardware o software specifici;

Unità ottiche

Il disco ottico è un tipo di supporto di memoria. È costituito da un disco piatto e sottile in genere di policarbonato trasparente. Al suo interno è inserito un sottile foglio metallico, in genere di alluminio, su cui vengono registrate e lette le informazioni tramite un raggio laser.

L'informazione su un disco ottico è memorizzata sequenzialmente in una traccia continua a spirale, dalla traccia più interna a quella più esterna. I dischi ottici sono particolarmente resistenti agli agenti atmosferici. Dal 2005 in poi, il loro utilizzo è andato progressivamente calando, a causa di altri dispositivi più compatti e veloci come hard disk portatili o pen drive. Esistono diversi tipi di dischi ottici:

CD-ROM: • Capienza: 700 MB • Velocità minima di lettura: 150 kB/s **DVD-ROM:** • Capienza: 4,7 - 17 GB • Velocità minima di lettura:1350 kB/s **Blue Ray:** • Capienza 25 - 100 GB • Velocità minima di lettura: 36 MB/s	

Periferiche di I/O

Le periferiche sono dispositivi hardware del computer che permettono all'utente di interagire con il computer. Si chiama periferica poiché il dispositivo è generalmente un componente esterno della scheda madre che contiene la CPU ed è collegato tramite cavo o tramite una connessione wireless. La periferica del computer è un dispositivo elettronico che viene collegato al computer per ricevere o inviare i dati, rispettivamente periferica (o dispositivo) di input e di output.

Nell'architettura hardware del computer le periferiche gestiscono il flusso di dati in entrata (input) e in uscita (output) del computer. Tutte le periferiche sono collegate alla CPU, a cui spetta il compito di controllare il flusso di dati da e verso le periferiche.

Le periferiche di un computer sono classificate in:

Periferiche di input: Le periferiche di input consentono all'utente di inserire dati o di richiedere l'esecuzione di un comando o di un programma, di selezionare una funzione, ecc. Sono unità periferiche di entrata. Il flusso di dati si muove dall'utente verso il computer. Alcuni esempi di periferiche di input sono:

- Mouse
- Tastiera
- Joystick
- Scanner
- Microfono

Periferiche di output: Le periferiche di output permettono all'utente di osservare il risultato dell'elaborazione dati. Sono unità periferiche di uscita. Il flusso di dati si muove dal computer verso l'utente. Alcuni esempi di periferiche di output sono:

- Monitor
- Stampante
- Casse audio
- Auricolari

Periferiche di input/output: Le periferiche di input/output (I/O) possono svolgere sia operazioni di lettura dei dati (input) che operazioni di scrittura dei dati (output). Alcuni esempi di periferiche I/O sono:

- Penne USB
- Schermi touchscreen
- Modem/Router

Bus

Il bus è un canale di comunicazione che permette a periferiche e componenti interni di un computer di interfacciarsi tra loro scambiandosi informazioni o dati di vario tipo attraverso la trasmissione e la ricezione di segnali.

Scheda madre

Le componenti hardware del computer sono progettate intorno ad una scheda principale che prevede alcuni circuiti integrati e molti componenti elettronici come i condensatori, le resistenze, ecc. Tutti questi componenti sono saldati sulla scheda e sono collegati dalle connessioni del circuito stampato e da un gran numero di connettori: questa scheda è detta scheda madre.

Sulla scheda madre sono presenti l’alloggiamento per il processore (chiamato anche socket), gli slot per installare la memoria RAM, gli slot per la scheda e per eventuali schede di espansone, i connettori per le porte USB ed i connettori per collegare le memorie di massa come hard disk, SSD, o unità DVD.

Inoltre, possono comprendere un certo numero di periferiche multimediali e di rete che possono essere disattivate:

- Scheda video integrata;
- Scheda audio integrata;
- Scheda di rete integrata.


M.2
ASUS
PRIME H670-PLUS D4

Case

Il case indica il contenitore dentro cui sono montati i componenti principali di un computer.

La scheda madre viene montata all’interno di un case che al suo interno prevede anche degli alloggiamenti per i dispositivi di memoria di massa (come hard disk o unità DVD), esternamente sono presenti dei tasti che permettono di accendere o riavviare il computer e un certo numero di luci (o led) che permettono di verificare lo stato di funzionamento dell'apparecchiatura e l'attività dei dischi rigidi. Nella parte posteriore, il case prevede delle aperture al livello delle schede di estensione oltre agli spazi dedicati alla scheda madre stessa per le interfacce (porte) dedicate a tastiera, mouse, stampanti, ecc..

Infine, il case ospita un blocco di alimentazione elettrica (detto comunemente alimentatore), che deve fornire una corrente elettrica stabile e continua a tutti gli elementi che costituiscono il computer. L'alimentazione serve quindi a convertire la corrente alternativa della rete elettrica (220 Volt) in tensione continua di 5 Volt per i componenti del computer e a 12 Volt per alcune periferiche interne (dischi, lettori CDROM, ecc.). Il blocco d'alimentazione è caratterizzato dalla sua potenza, che condiziona il numero di periferiche che il computer è capace di alimentare. La potenza del blocco d'alimentazione è generalmente compresa tra 250 e 800 Watt.

3. “Software”

Definizione di software

Il software ha l’importante funzionalità di fare da tramite tra l’utente e l’hardware.
In questo modo l’utente interagisce direttamente (a livello logico-funzionale) con il software e, di conseguenza, indirettamente (a livello fisico) con l’hardware. L’utente è così indipendente dalla conoscenza dei complessi meccanismi che lo caratterizzano.

In pratica, un software è la parte logica di un elaboratore ed è formato da sequenze di istruzioni che guidano l'hardware nell'esecuzione dei suoi compiti, e che definiscono inequivocabilmente l'esecuzione di un determinato compito. Genericamente per software si intende l’insieme dei programmi impiegati in un sistema di elaborazione dati che gestisce il funzionamento di un elaboratore.

Si divide in due componenti principali:

- Software di base (sistemi operativi);
- Software applicativi (programmi o app)

Software di base

Il software di base, conosciuto anche come software di sistema, è il software in grado di gestire tutte le risorse hardware di un qualunque computer. In parole povere, è la parte del software più vicina all’hardware della macchina. Parlando sommariamente, il software di base si divide in tre diverse categorie:

- i sistemi operativi, tipo Windows, Linux, Android o macOS;
- i compilatori e gli interpreti;
- e le librerie.

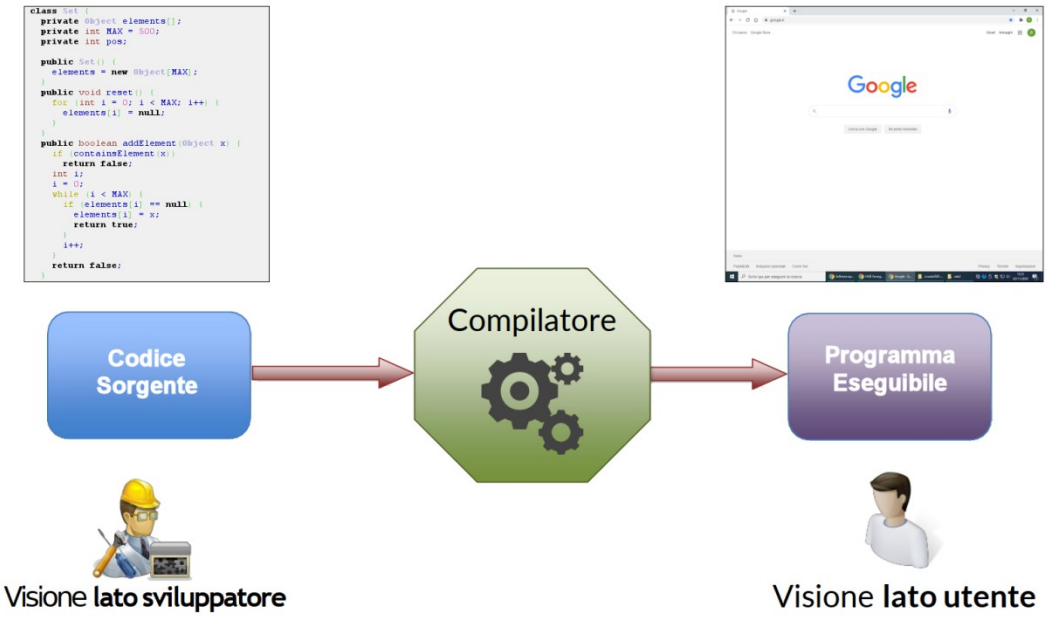
Software applicativo

Il software applicativo, conosciuto anche con il termine di applicazione (o app), è l’insieme dei programmi che aiutano l’utente a risolvere una vasta tipologia di problemi. In altre parole, si tratta di tutte quelle applicazioni che non fanno parte del sistema operativo ma che sono comunque necessarie all’utente per rendere possibile una o più determinate funzionalità.
Fanno parte di questa categoria la quasi totalità dei software (ad esclusione dei sistemi operativi), una sommaria suddivisione può essere fatta in base all’uso a cui è destinato, fermo restando che ogni categoria può essere suddivisa in tante altre sotto-categorie e così via:

- **software di produttività**: di questa categoria fanno parte, ad esempio, gli elaboratori di testi (word processor), i fogli di calcolo, i software per le presentazioni, gestione di basi di dati (database), i software di grafica e fotoritocco, client di posta elettronica, ecc.;
- **software di svago**: videogiochi per computer o console, emulatori, lettori audio e video;
- **software educativo:** cioè tutte quelle applicazioni utili alle scuole di qualsiasi grado;
- **software scientifici**: ossia tutte quelle applicazioni utili nei vari campi della scienza applicata, tipo astronomia, biologia o chimica;
- **software di sviluppo (software development):** comprende programmi e applicazioni che gli sviluppatori utilizzano per creare, per rilevare i bug (debug) o per modificare altro software, come ad esempio i cosiddetti IDE;
- **software gestionali**: tutte quelle applicazioni studiate principalmente per le aziende, come gestione del personale, magazzino, fatturazione, processi decisionali, CRM, ERP, OLAP, project management ed e-commerce, ecc.;

- **utilities**: è una tipologia di software che può essere utilizzata per l'analisi, la configurazione, l'ottimizzazione e/o la manutenzione del computer. La sua funzione è eseguire la configurazione e/o il monitoraggio di dispositivi hardware o software. Un esempio possono essere: gli antivirus, gestori degli appunti, programmi diagnostici, gestori di pacchetti, ecc.

Per realizzare un software il programmatore deve scrivere il **codice sorgente** in un determinato linguaggio di programmazione (Java, C++, Python, ecc.). Il codice sorgente deve essere tradotto in linguaggio macchina per poter essere eseguito, ovvero per poter essere trasformato in un programma eseguibile (questa operazione viene effettuata a sua volta da un programma, chiamato **compilatore**). Il software viene venduto spesso sotto forma di programma eseguibile, prima di tutto per evitare che possa essere modificato e rivenduto.



In realtà i software possono essere anche classificati in base a diverse loro caratteristiche che possono essere svariate, ecco alcuni esempi:

- **licenza** (software libero o software proprietario);
- **sistema operativo su cui possono essere utilizzati** (Windows, Android, Linux, ecc.);
- **da installare** o **portabile**;
- **stand alone** (ovvero che possono girare completamente autonomi su sistemi isolati) oppure **network** (ovvero che funzionano in un ambito di rete).

Licenze e distribuzione del software

Una licenza d'uso, in informatica, è il contratto con il quale il titolare dei diritti di sfruttamento economico sul software definisce il regime giuridico di circolazione e le limitazioni nell'utilizzo.
Quando si installa un software nella maggior parte dei casi è necessario accettare la licenza d'uso prima di poterlo usare.
È' fondamentale leggere attentamente la licenza prima di utilizzare qualsiasi app poiché la sua accettazione pone l'utente in regola rispetto alle norme sul copyright[1], infatti in un software

[1] Il copyright (termine di lingua inglese che letteralmente significa diritto di copia), identifica il diritto d'autore su una determinata opera di qualsiasi natura, che implica il divieto di ogni riproduzione e vendita abusiva per un determinato

proprietario la licenza consente al beneficiario il suo utilizzo sotto particolari condizioni ed impedendone altre come lo studio, la modifica, la condivisione, la ridistribuzione o l'ingegneria inversa. Le restrizioni sono imposte dal titolare dei diritti di sfruttamento economico, (l'autore o il cessionario dei diritti in questione), tramite mezzi primariamente giuridici, come costo di utilizzo, installazione su un numero ristretto di dispositivi, o la circolazione del codice sorgente.
In pratica quando si acquista un software, non si diventa proprietario ma si acquista l’autorizzazione ad usarlo (con le eventuali limitazioni).

Classificazione delle licenze

I software possono essere classificati in base a diverse loro caratteristiche, in particolare in base alla loro licenza, esso può essere suddiviso in 2 categorie:

- software proprietario (closed source)
- software libero (open source)

Software proprietario

Il software proprietario è un software protetto dalle leggi sul copyright e in alcuni casi anche dal brevetto, distribuito attraverso la sottoscrizione di una licenza d’uso, in genere a pagamento.

Software libero

Il software libero (chiamato anche open source), prevede un uso a titolo gratuito grazie ad una licenza copyleft[2]. Un software open source è reso tale per mezzo di una licenza attraverso cui i detentori dei diritti favoriscono la modifica, lo studio, l'utilizzo e la redistribuzione del codice sorgente.
Le più diffuse licenze sono:

- **GNU GPL (GNU General Public License):** è una licenza copyleft per software libero, Fondamentalmente stabilisce che un'opera protetta da GNU GPL deve rimanere libera, ovvero col susseguirsi delle modifiche deve continuare a garantire ai suoi utenti le cosiddette quattro libertà:
    1. Libertà di eseguire il programma per qualsiasi scopo;
    2. Libertà di studiare come funziona il programma e di modificarlo in base alle proprie necessità;
    3. Libertà di ridistribuire copie del programma in modo da aiutare il prossimo;
    4. Libertà di migliorare il programma e di distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio.
- **CC (Creative Commons)** che permette la copia e la distribuzione di software purché ciò non avvenga a scopo di lucro.

Il software viene realizzato da aziende chiamate software house ed è un’opera protetta dal Decreto Legislativo n. 518 del 29/12/1992. Secondo la Legge n. 633 del 22/04/1941 (sul copyright) l’autore di un programma ha il diritto esclusivo di effettuare o autorizzare:

numero di anni stabilito dalla legge.

[2] Il copyleft (talvolta indicata in italiano con "permesso d'autore"), indica un modello di gestione dei diritti d'autore basato su un sistema di licenze attraverso le quali l'autore (in quanto detentore originario dei diritti sull'opera) indica ai fruitori dell'opera che essa può essere utilizzata, diffusa e spesso anche modificata liberamente, pur nel rispetto di alcune condizioni essenziali. Può essere applicato ad una moltitudine di opere anche al di fuori dell’informatica.

- la riproduzione permanente o temporanea, totale o parziale, del programma per elaboratore con qualsiasi mezzo o in qualsiasi forma;
- la modifica del programma, oltre alla traduzione e all’adattamento;
- la distribuzione del programma originale o di copie dello stesso.

4

4. “Elaborazione digitale di immagini e suoni”

Introduzione

L'elaborazione digitale delle immagini è una attività che, mediante l'utilizzo di specifici programmi e dispositivi elettronici come computer e tablet, permette di modificare un'immagine digitale apportando variazioni visibili e apprezzabili all'osservatore. È una attività comune nel campo fotografico professionale.

Inoltre, ci si riferisce anche a tutte le operazioni eseguite su un’immagine per trasformarla, in modo da rendere più agevole l’estrazione di informazioni riguardanti gli oggetti in essa contenuti, per riprodurla o trasmetterla.

Le immagini sono classificabili a seconda del metodo con cui esse vengono generate, come ad esempio fotografie, disegni, dipinti, immagini televisive, ecc.

Da tutti i tipi di immagini si possono ottenere le corrispondenti immagini numeriche o digitali generando funzioni f(x,y) che misurano l’intensità luminosa nei punti di coordinate x,y dell’immagine stessa.

Un’immagine digitale non è visibile direttamente. È un concetto che acquista il suo significato quando è possibile visualizzarla con un’attrezzatura adeguata. In pratica i dati che formano un’immagine digitale devono essere restituiti, dopo una elaborazione, su uno schermo.

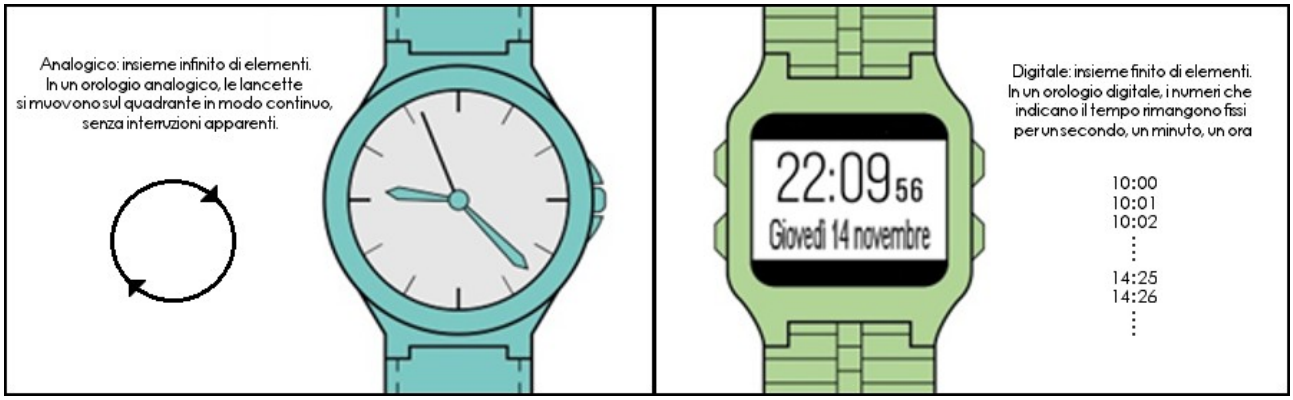
Analogico e digitale

Quando si parla di computer, tablet, smartphone si parla sempre di “digitale” e, qualche volta, si contrappone a questo termine quello di “analogico”.

“**Analogico**” e “**digitale**” sono termini che si incontrano di continuo quando si parla di tecnologie (vecchie e nuove). Nel senso comune, ad “analogico” è associato un significato di “vecchio” o

“passato” o “di bassa qualità”; mentre “digitale” è, invece sinonimo di “nuovo” o “innovativo” o “di qualità”. Questa distinzione da senso comune NON è vera.

Prima di tutto va tenuto presente che quando si parla di analogico e di digitale ci si riferisce alle modalità di rappresentare la misura di una quantità, a modi in cui variano le grandezze che si vuole prendere in considerazione (come un segnale audio, una immagine, un segnale video, il colore, ….).



Per **analogico** si intende delle grandezze che assumono valori in un insieme continuo, in particolare una variabile analogica può assumere un numero infinito di valori (ad esempio la distanza tra due punti nello spazio può assumere un numero infinito di valori).

Per **digitale** si intende delle grandezze che assumono valori all’interno di un insieme di dimensioni discrete, in particolare una variabile digitale può assumere solo un numero finito di valori (la durata di un giorno – ad esempio, può assumere solo uno dei 3.600 valori se usiamo i minuti , oppure uno degli 86.400 valori se usiamo l’unità “secondo”, oppure uno degli 864.000 valori se usassimo i decimi di secondo o uno degli 8.640.000 valori se si usano centesimi di secondo; tante possibilità ma pur sempre finite, determinate).

Tipi di immagini digitali

Esistono due tipi di immagini digitali: le immagini raster (o bitmap) e le immagini vettoriali. Le prime sono costituite da una “mappa” di bit, mentre le seconde sono il risultato di funzioni matematiche.

Immagini raster (o bitmap)

Le immagini raster sono chiamate anche immagini **bitmap**, in questo tipo di immagini, i valori memorizzati indicano le caratteristiche di ogni punto dell'immagine da rappresentare (pixel). Nelle immagini a colori, viene memorizzato solitamente il livello di intensità dei colori fondamentali (nel modello di colore RGB, uno dei più usati, sono tre: rosso, verde e blu. Un altro esempio è CMYK, usato per la stampa, basato su quattro colori fondamentali: ciano, magenta, giallo e nero) mentre nelle immagini monocromatiche in scala di grigio (dette impropriamente bianco e nero) il valore indica l'intensità del grigio, che varia dal nero al bianco

Vantaggi della grafica raster

- Ricca gradazione di colori e luminosità;
- Si può modificare ogni singolo punto.

Svantaggi della grafica raster

- Perdita di qualità quando si scala (ingrandisce) l‘immagine;
- La compressione può portare a una perdita di qualità;
- È richiesto molto spazio per memorizzare un’immagine di qualità;
- La vettorizzazione comporta un grande dispendio computazionale.

Immagini vettoriali

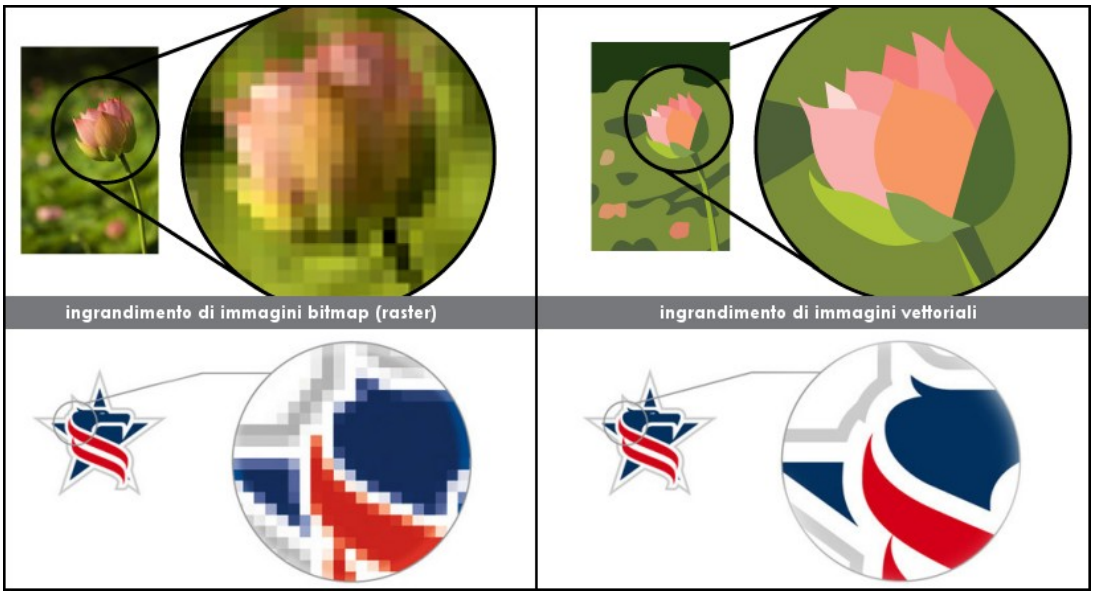
Una immagine vettoriale è descritta come un insieme di primitive geometriche (punti, linee, segmenti, poligoni, ecc.) alle quali possono essere attribuiti colori e anche sfumature.
Queste immagini presentano due grandi vantaggi: possono essere ridimensionate senza perdita di qualità e occupano poca memoria.
In altre parole, si può dire che questo tipo di immagini sono ottenute dall'unione di un certo numero di punti o nodi, che formano linee e poligoni, a loro volta uniti in strutture più complesse, fino a formare l'immagine voluta.
Questo tipo di immagine è utilizzato nel disegno tecnico per la progettazione architettonica ed industriale, nella rappresentazione dei caratteri negli elaboratori di testo, nella grafica per la creazione di loghi e marchi o altri oggetti, ecc...

Vantaggi della grafica vettoriale

- Scalabile senza perdite di qualità;
- Compressione senza perdite di qualità;
- Dimensioni del file ridotte;
- Facile gestione delle eventuali modifiche;
- La conversione a immagine raster è semplice;
- La grafica vettoriale, essendo definita attraverso equazioni matematiche, è indipendente dalla risoluzione, mentre la grafica raster, se viene ingrandita o visualizzata su un dispositivo dotato di una risoluzione maggiore di quella del monitor, perde di definizione.

Svantaggi della grafica vettoriale

- Non adatta a rappresentazioni grafiche complesse;
- La realizzazione di immagini vettoriali non è una attività intuitiva come nel caso delle immagini raster;
- Utilizzo di strumenti avanzati per creare immagini vettoriali complesse;
- Risorse adeguate alla complessità dell’immagine: una immagine vettoriale molto complessa può essere molto corposa e richiedere l'impiego di un computer molto potente per essere elaborata.



Elementi di una immagine raster

In una immagine digitale raster gli elementi base sono:

- **Pixel:** unità elementare dell’informazione visualizzata sullo schermo di un computer

- **Risoluzione grafica:** Indica la densità lineare di un’immagine
- **Risoluzione dello schermo**: Indica la grandezza (base per altezza), ed è espressa in Pixel
- **Profondità di colore**: Indica il numero di bit usati per indicare il colore di un singolo pixel.
- **Aspect ratio**: Indica il rapporto tra la larghezza e l'altezza (sempre espressi in pixel) dell'immagine digitale.

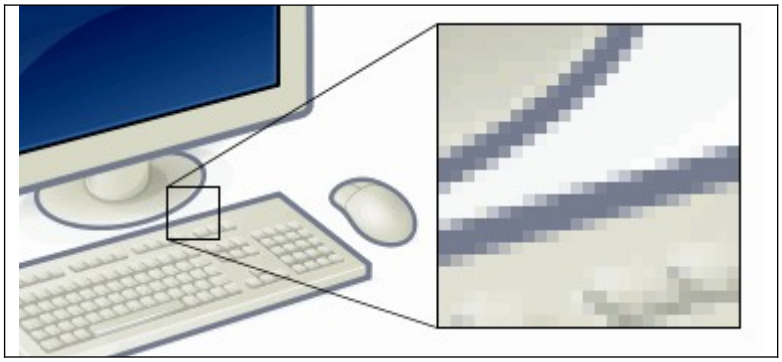
In una immagine raster possono essere effettuate delle operazioni come:

- **Elaborazione:** In cui le immagini digitali possono subire un processo di elaborazione volto a manipolare/modificare l'immagine stessa secondo le caratteristiche desiderate (es. fotoritocco, restaurazione delle immagini ecc..).
- **Compressione:** Come caso particolare di elaborazione digitale delle immagini, le immagini digitali possono a loro volta subire, secondo vari possibili standard, un processo di compressione dell'immagine volto a ridurre la quantità di informazione associata per la trasmissione sul canale di comunicazione o la memorizzazione su supporti di memorizzazione (in parole povere al fine di occupare meno spazio mantenendo o meno una certa qualità)

Pixel

Tutti i dispositivi di output riproducono immagini mediante i pixel (abbreviativo di “picture elements” o in italiano “elemento di immagine”), che sono unità elementari di visualizzazione su schermo (in pratica rappresenta l'unità minima convenzionale della superficie di un'immagine digitale). Essi sono disposti in modo da comporre una griglia fissa rettangolare, che per la loro piccolezza e densità appaiono fusi in un'unica immagine.

Semplificando il tutto, si può dire che i pixel sono dei piccoli quadratini che rappresentano un singolo colore e che, vicini tra di loro, compongono l’immagine, similmente ad un mosaico.



Il colore di ogni singolo pixel viene definito come una combinazione dei tre colori primari: blu, rosso, verde. Questo non è l'unico modo di definire un colore, esistono altri modi che vengono chiamati spazi di colore, ma nel caso delle immagini generate al computer il sistema RGB (RED Rosso, GREEN Verde, BLUE Blu) è il più diffuso dato che le schede grafiche lo utilizzano nativamente per generare il segnale da visualizzare con il monitor.

Risoluzione grafica

La risoluzione grafica è la grandezza che quantifica il grado di dettaglio di un'immagine, dato dal numero di punti immagine che la compongono linearmente (altezza o larghezza).

La risoluzione di un'immagine raster dipende dalla densità dei pixel che la compongono. Il concetto di risoluzione (intesa come densità) ha senso quando si parla della riproduzione di un oggetto come, ad

esempio, l'immagine di una fotografia o la scansione di una pagina scritta o disegnata, ma perde di significato quando viene riferita a un file totalmente generato da un programma di grafica vettoriale.

Il livello di dettaglio di un'immagine digitale dipende dalla risoluzione, cioè dalla quantità di informazione che essa contiene. Ma per vedere la massima risoluzione dell'immagine, è necessario che anche il sistema di riproduzione sia in grado di mostrarla (Schermo, stampante, ecc.) e che abbia una dimensione reale adeguata a mostrarla ai nostri occhi. Per descrivere le varie densità di risoluzione è necessario fare alcune differenziazioni. Sebbene simili, le unità di misura della risoluzione "DPI", "PPI" si riferiscono a separati metodi di misurazione, nonostante nella pratica tendano a essere utilizzate anche intercambiandole.

DPI (Dots Per Inch): traduzione di “punti per pollice”. Quest’unità di misura indica il numero di punti che si trovano in un pollice[3]. Il termine DPI viene utilizzato principalmente nel mondo della stampa, si riferisce quindi ai dispositivi di output come stampanti a getto di inchiostro o stampanti laser. Esso indica la definizione di stampa e stabilisce il numero di punti di inchiostro stampati per ogni pollice; di conseguenza, maggiori saranno i DPI e maggiore sarà dettagliata l’immagine stampata in quella porzione di foglio.

PPI (Pixel Per Inch): traduzione di “pixel per pollice”. Quest’unità di misura si riferisce ai dispositivi di input come fotocamera o scanner, e rappresenta la quantità di pixel all’interno di un pollice che un sensore fotografico è in grado di scansionare e campionare.

[3] Un pollice equivale a circa 2,54 cm

Risoluzione dello schermo

Il numero di pixel sul monitor è definito dalla sua risoluzione, essa è espressa nella dimensione orizzontale ed in quella verticale. Perciò, un monitor con una risoluzione di 1920 x 1080, ha 1920 pixel nella dimensione orizzontale e 1080 nella verticale. Fondamentalmente, la risoluzione determina la qualità dell’immagine, maggiori saranno i pixel e più sarà definita l’immagine.

Principali risoluzioni usate in ambito informatico e televisivo:

Standard Video	Risoluzione	Aspect Ratio	Profondità del colore
VGA	640x480	4:3	4 bpp
SVGA	800x600	4:3	4 bpp
XGA	1024x768	4:3	8 bpp
HD	1280x720	16:9	24 bpp
Full HD	1920x1080	16:9	24 bpp
4K	3840x2160	16:9	24 bpp
8K	7680x4320	16:9	30 bpp
16K	15360x8640	16:9	30 bpp

Si tratta solo di un esempio delle varie risoluzioni possibili, non vanno imparate a memoria.

Alcuni computer adottano risoluzioni completamente diverse, ad esempio sul macbookpro 2015, 15 pollici, con scheda combinata Intel + Nvidia, la risoluzione dello schermo é:

2880x1800

Sullo stesso modello, ma con schermo da 13 pollici, la risoluzione è:

2560x1600

Come vedete nessuno di questi due dispositivi adotta una risoluzione standard, il che è perfettamente lecito.

Profondità di colore

La profondità di colore indica il numero di bit usati per indicare il colore di un singolo pixel.
Quando si fa riferimento a un pixel, il concetto può essere definito come il numero di bit per pixel (bpp), che specifica il numero di bit impiegati.
Fino alla fine degli anni 90, con le profondità di colore più basse, il valore memorizzato per ciascun bit era generalmente un indice in una mappa di colori o tavolozza[4]. I colori disponibili nella tavolozza stessa possono essere determinati dall'hardware oppure modificabili.
Era consuetudine indicare i diversi tipi di profondità con il nome della norma video con il quale sono stati introdotti sul mercato dei personal computer.

- 1 bpp (2^1 = 2 colori): grafica monocromatica, (in bianco e nero)
- 2 bpp (2^2 = 4 colori): grafica CGA
- 4 bpp (2^4 = 16 colori): grafica EGA o VGA standard a bassa risoluzione
- 8 bpp (2^8 = 256 colori): grafica VGA ad alta risoluzione, Super VGA

Con l'aumentare del numero di bit per pixel aumenta anche la quantità di colori possibili, rendendo sempre più scomodo l'uso delle tavolozze. Per le profondità più alte si preferisce perciò codificare i colori direttamente nei valori corrispondenti alla luminosità relativa dei canali rosso, verde e blu secondo il modello RGB.

- 16 bpp (2^{16} = 65. 536 colori): High color
- 24 bpp (2^{24} = 16.777.216 colori): Truecolor

Il modello di profondità truecolor utilizza 24 bit e permette di riprodurre immagini in modo molto fedele alla realtà arrivando a rappresentare 16,8 milioni di colori distinti. Con questa profondità si usano 8 bit per rappresentare il rosso, 8 bit per rappresentare il blu ed 8 bit per rappresentare il verde. I 28 = 256 livelli d'intensità per ciascun canale si combinano per produrre un totale di 16.777.216 colori (256 × 256 × 256).

Di seguito vengono riportate 2 immagini: sulla sinistra abbiamo un’immagine con una profondità di colore di 24 bpp, mentre a destra la stessa immagine con una profondità di colore a 4 bpp.


Anche per le fotografie digitali, vale lo stesso discorso fatto in precedenza, infatti anche le foto, essendo delle immagini digitali, sono composte da un certo numero di pixel.

Nelle attuali fotocamere, spesso la risoluzione massima viene indicata in megapixel (MP). Per fare un esempio concreto, una fotocamera che arriva a 16 MP, con foto in formato 16:9 avrà una risoluzione è di 5.312 pixel in larghezza e 2.988 in altezza; moltiplicando tali valori si ottiene un’area di 15.872.256 pixel, vicina appunto ai 16 MP nominali.

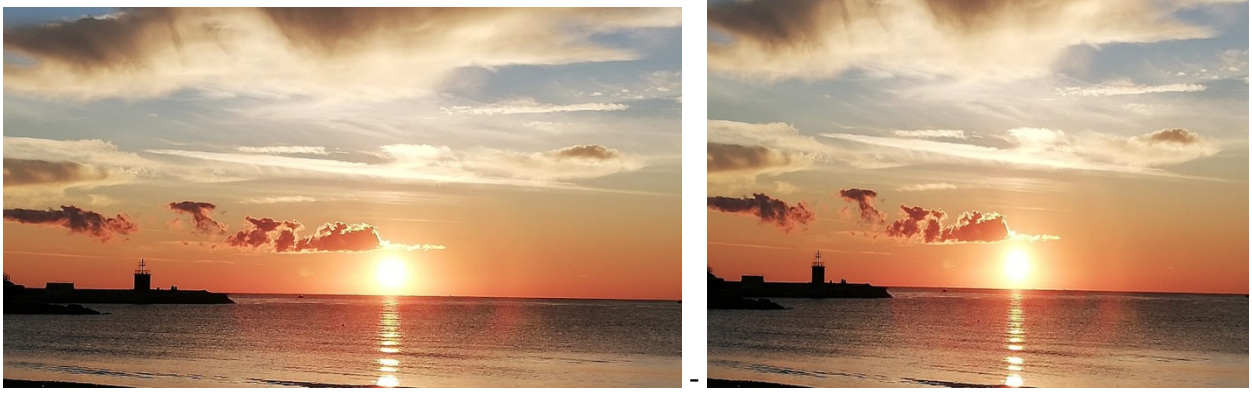
Ovviamente indichiamo megapixel perché parliamo appunto di milioni di pixel, se ipotizziamo una risoluzione come ad esempio 50.000 x 50.000 avremo 2.500.000.000 di pixel e quindi potremo indicarli anche come 2,5 gigapixel (GP).

[4] Una tavolozza indica un insieme dei colori disponibili in un programma di grafica o di disegno.

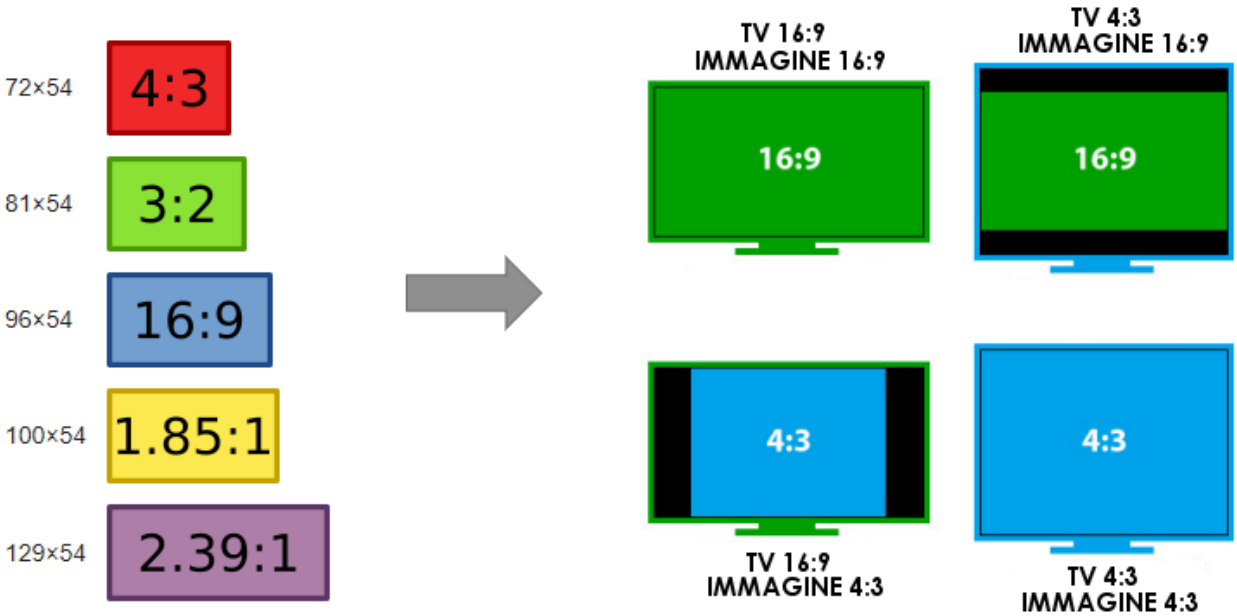
Aspect Ratio

Un'altra caratteristica importante è il rapporto d'aspetto (o aspect ratio), ovvero il rapporto tra la larghezza e l'altezza (sempre espressi in pixel) dell'immagine digitale.

Questo rapporto dipende anche dal dispositivo usato per rappresentare l'immagine: su alcuni dispositivi i pixel hanno una forma rettangolare, mentre se viene modificato l'aspect ratio di un'immagine otterremo una distorsione, come possiamo notare dalla figura qui sotto:



Per esempio, un'immagine con risoluzione 1920 x 1080, quindi con aspect ratio 16:9 visualizzata su un monitor con aspect ratio 4:3 risulterà distorta oltre che schiacciata orizzontalmente.



In figura, a sinistra cinque proporzioni comunemente usate (Rappresentate tutte con la stessa altezza). Mentre a destra, adattamento di formati differenti di immagini in tv/monitor con uguale o differente formato.

Codice esadecimale dei colori

Questo tipo di notazione prende il nome di codice esadecimale dei colori o **codice HTML**.

Generalmente, nella maggior parte dei programmi di grafica per identificare inequivocabilmente un colore ben specifico, sono indicati con numeri esadecimali. I numeri esadecimali sono numeri in base 16 e più precisamente sono: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, dove zero è pari a 0 e F è paria a 15. Il numero esadecimale con valore più alto è FF che equivale al valore decimale 255.

I colori sono indicati con un codice a 6 caratteri preceduto dal simbolo # che indica appunto un colore. I codici dei colori sono triplette esadecimali (e quindi una cifra composta da 6 cifre esadecimali), che rappresentano i colori rosso, verde e blu (i tre colori primari).

La base esadecimale è usata anche dai vari browser per navigare su internet, per identificare univocamente un colore ben specifico.

Si prenda, ad esempio, il colore bianco, in base esadecimale è indicato come #FFFFFF è composto da:

- FF ossia Rosso = 255
- FF ossia Verde = 255
- FF ossia Blu = 255

255 parti di rosso, 255 parti di verde e 255 parti blu

Di seguito alcuni esempi di colori con il codice esadecimale corrispondente.

NOME	CODICE	COLORE
Bianco	#FFFFFF	
Nero	#000000	
Rosso	#FF0000	
Blu	#0000FF	
Verde	#00FF00	
Giallo	#FFFF00	
Magenta	#FF00FF	
Celeste	#00FFFF	
Viola	#5C2E91	
Marrone	#A56A1B	
Beige	#E3DAC9	
Azzurro	#0080FF	

Compressione delle immagini

Concorrono a determinare la dimensione dell’immagine:

- Risoluzione
- Profondità di colore
- Formato del file (Compresso o non compresso, che a sua volta può essere lossless o lossy)

La dimensione in byte di un’immagine varia a seconda della sua risoluzione e a seconda del numero dei colori che può assumere un pixel.

Assumendo che nelle tecniche di fotoritocco ed elaborazione delle immagini ha poco senso diminuire la profondità di colore, per ridurre la dimensione dell’immagine, questa deve essere compressa, cioè con tecniche matematiche si devono ridurre il numero di byte necessari per contenere l’immagine.

Esempio di compressione di un file:

> **AAABBBBAAAAAAAAACCCC**
> (codifica ASCII peso 20 byte)
>
> la stessa sequenza può essere rappresentata da
>
> 3A4B9A4C
> (peso 8 byte).

Formato dei file grafici

La grafica annovera una lunga lista di formati diversi, perché le immagini possono essere codificate e/o rappresentate in molti modi diversi: inoltre molti programmi di grafica che trattano immagini associano ai dati grafici veri e propri anche una serie di informazioni supplementari, per esempio sulla loro rappresentazione.

I file grafici hanno 2 tipi di rappresentazioni principali:

- raster : rappresentazione con matrice di pixel, ognuno con un valore di colore
- vettoriale : ogni elemento ha una forma definita matematicamente (retta, circonferenza, triangolo, ellisse, parallelogramma, ecc) in tutte le sue dimensioni, e 2 coordinate che ne indicano la posizione.

I 2 formati vengono usati per scopi diversi.

Il formato vettoriale é tipicamente usato in fase creazione di disegni, da zero, o quando e’ importante poter ingrandire o rimpicciolire (termine tecnico: ‘scalare’) l’immagine senza perdere in qualita’ della stessa.

Inoltre con il formato vettoriale le immagini hanno una dimensione in termini di kB, o Mb, molto ridotta.

Il formato raster é più usato quando si lavora a partire da immagini preesistenti, tipicamente fotografiche, e si vuole lavorare sui valori di colore dell’immagine in generale, o di una zona di essa, non andando amanipolare i singoli elementi grafici. Questo tipo di formato non preserva la qualita nelle operazioni di scalatura (ingrandimento e rimpicciolimento).

Con il formato raster le immagini hanno una dimensione, in termini di kB o mB, molto grande rispetto a i file vettoriali, perche bisogna rappresentare ogni pixel dell’immagine singolarmente.

Per ovviare a questo inconveniente, le immagini raster sono quasi sempre compresse.

Formati delle immagini con compressione

Esistono molte tecniche per la compressione dei dati e bisogna saper distinguere tra tecniche LOSSLESS e tecniche LOSSY.

Nelle prime la riduzione di spazio non comporta perdite di informazione ed è possibile tornare al formato originale.

Alcune di queste tecniche sono:

- **GIF**: (Graphic Interchange Format)[5]
- **PNG**: (Portable Network Graphics)
- **TIFF**: (Tagged Information File Format)

Le tecniche LOSSY invece prevedono una perdita del contenuto dell’immagine, ma tale da non creare perdita del significato dell’immagine stessa.

Il formato più diffuso è il **JPEG**: (Joint Photographic Expert Group) molto usato nel mondo di internet dove non è importante l’immagine completa ma il suo significato. Va però detto che compressioni troppo spinte causano una perdita eccessiva di definizione e di fedeltà dei colori dell’immagine.

[5] fino a 256 colori la compressione avviene senza perdita di informazioni, per immagini con più di 256 colori si ottiene una compressione lossy poiché vengono eliminate la maggior parte delle sfumature di colore

La compressione con perdita è spesso da preferire perché crea immagini significativamente più compresse, mantenendo una buona qualità.

Esistono comunque situazioni nelle quali utilizzare un'immagine approssimata non è accettabile:

- immagini mediche
- documenti storici
- immagini con valore legale

Formato delle immagini senza compressione

Esistono anche dei formati per le immagini raster che non sono compressi, questi formati di file hanno richieste di elaborazione minima, non essendo necessari algoritmi di compressione (in fase di scrittura) e decompressione (in fase di lettura), tuttavia, mancando di compressione, risultano particolarmente voluminosi, in termini di spazio occupato su disco (o altro dispositivo di memorizzazione), rispetto agli altri formati:

Alcuni di questi formati sono:

- **BMP** (windows bitmap)
- **Raw** (immagine grezza, esistono molti file Raw secondo formati proprietari)

Formato BMP

Windows bitmap è un formato di file per immagini digitali di tipo bitmap (detto anche tipo raster) utilizzato nella grafica digitale.

L'estensione ***.bmp** definisce i file d’immagine di questo tipo (anche se meno frequentemente è usato il formato *.dib)

Una delle caratteristiche essenziali del formato bitmap, che ne hanno fatto per molto tempo la fortuna, è la velocità con cui le immagini vengono lette o scritte su disco, molto maggiore se paragonata a quella di altri tipi di file, soprattutto sulle macchine più lente. Nelle bitmap, essendo non compresse, la rappresentazione dei dati nella memoria RAM è in gran parte simile, spesso identica, a quella dei dati su disco: il processore non è costretto ad effettuare calcoli laboriosi durante le operazioni di codifica e di decodifica e il tempo di accesso ai dati è spesso limitato solo dall'hardware del drive.

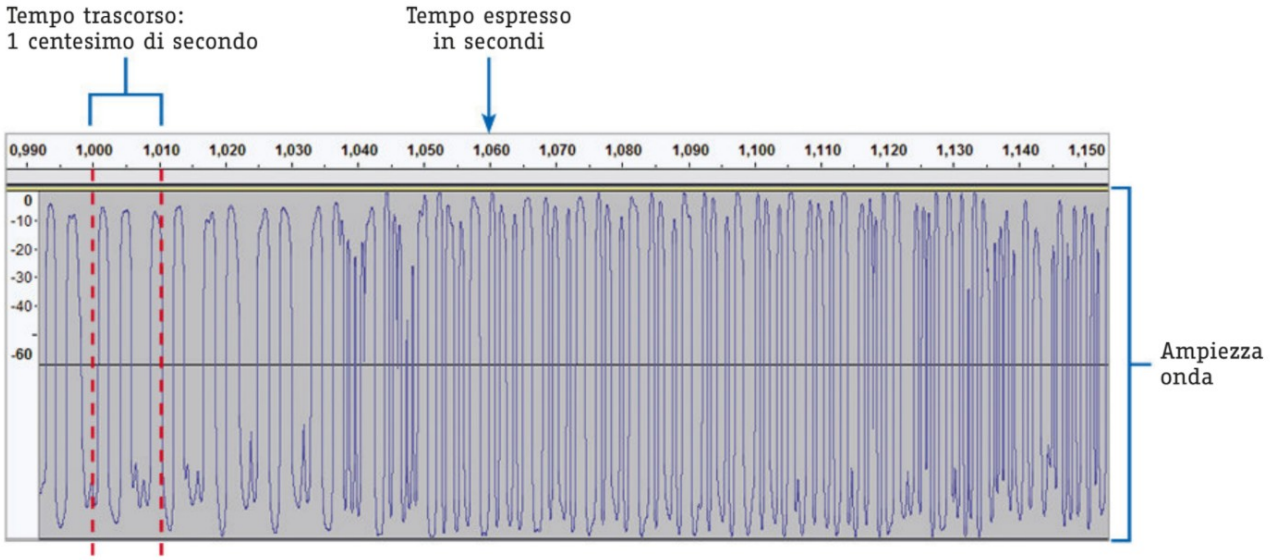
DEFINIZIONE:	Formato grafico senza compressione di tipo raster (non vettoriale)
COMPRESSIONE:	Il formato BMP non utilizza nessun tipo di compressione.
APPLICABILITÀ:	Le immagini mediche, quali per esempio quelle prodotte dalla TAC o dalla RM (Risonanza Magnetica) sono codificate in un formato tipo bitmap. Le immagini che hanno un valore legale, come rivendicazione del copyright o informatica forense.

Nella elaborazione delle immagini, le bitmap si rivelano adatte soprattutto alla memorizzazione temporanea delle immagini che vengono modificate spesso. Molti software di scansione per Windows salvano le immagini digitalizzate per default come file bitmap.

I suoni digitali

Lo onde sonore prodotte dai diversi suoni che ci circondano provocano oscillazioni della pressione in tempi brevissimi. La figura seguente mostra l'onda sonora emessa dalla voce umana, registrata grazie a un software di gestione audio.

Il grafico mostra sull’asse delle X il tempo in secondi e sull'asse delle Y il guadagno in dB. Possiamo notare come le onde di variazione oscillino in tempi bassissimi.



Dal punto di vista fisico i suoni che udiamo sono onde di pressione dell'aria. La pressione atmosferica a livello del mare è pari a 1013 mbar circa, i suoni non sono altro che perturbazioni della pressione caratterizzate da variazioni molto rapide.

I trasduttori sono dispositivi che trasformano le onde di pressione in oscillazioni di altro genere. Un tipico esempio di trasduttore audio è il microfono che “traduce" le oscillazioni della pressione in oscillazioni di segnale elettrico. Un altro esempio di trasduzione è rappresentato dalle ondulazioni presenti nei solchi dei dischi in vinile, che variano in maniera direttamente proporzionale rispetto alle onde di pressione. Entrambi questi sistemi (microfono, disco in vinile) utilizzano segnali analogici. I sistemi di tipo analogico, pur essendo molto fedeli nella registrazione, introducono notevoli quantità di rumore.

Campionamento di suoni

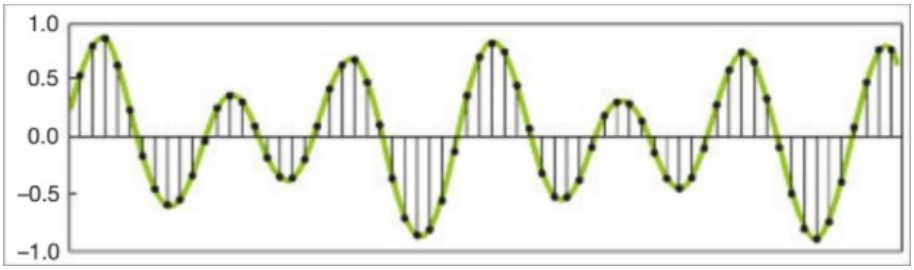
Per convertire un segnale analogico in digitalo, si eseguono campionamento e quantizzazione. Il segnale analogico, continuo nel tempo e in ampiezza, viene campionato, cioè misurato in intervalli di tempo costanti, ottenendo un sognale discreto nel tempo e continuo in ampiezza; poi viene quantizzato, ottenendo come risultato un segnale digitale discreto sia nel tempo che in ampiezza.

La scheda audio dei computer esegue da 8.000 a 48.000 campionamenti al secondo (frequenze di campionamento da 8 a 48 kllz), quantizzando poi tali dati su 8. 10 o 32 bit.

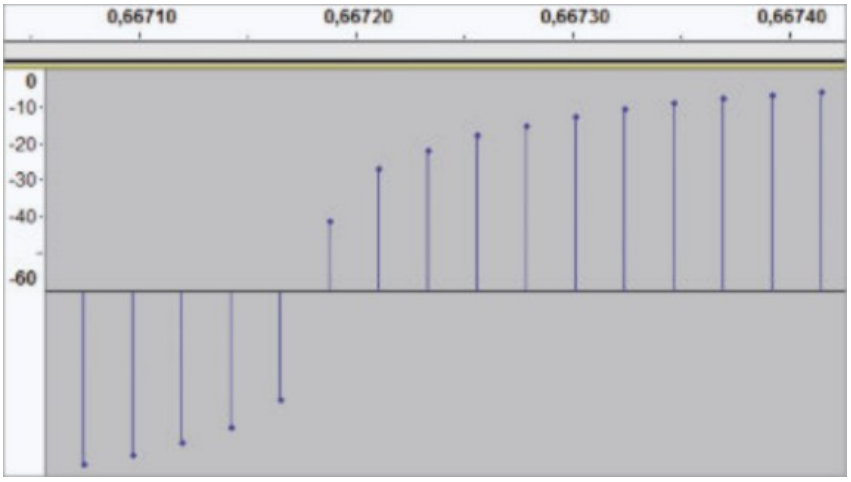
Nel caso dei suoni si parla di campioni temporali, mentre per le immagini di campioni spaziali.

Per ricostruire nel modo più corretto possibile tutte le forme d'onda da campionare, la frequenza di campionamento dovrebbe essere almeno il doppio di queste ultime, come specificato dal **teorema di**

Shannon. Nella figura è mostrata una forma d'onda (verde) che viene campionata (puntini neri) in istanti di tempo costanti.

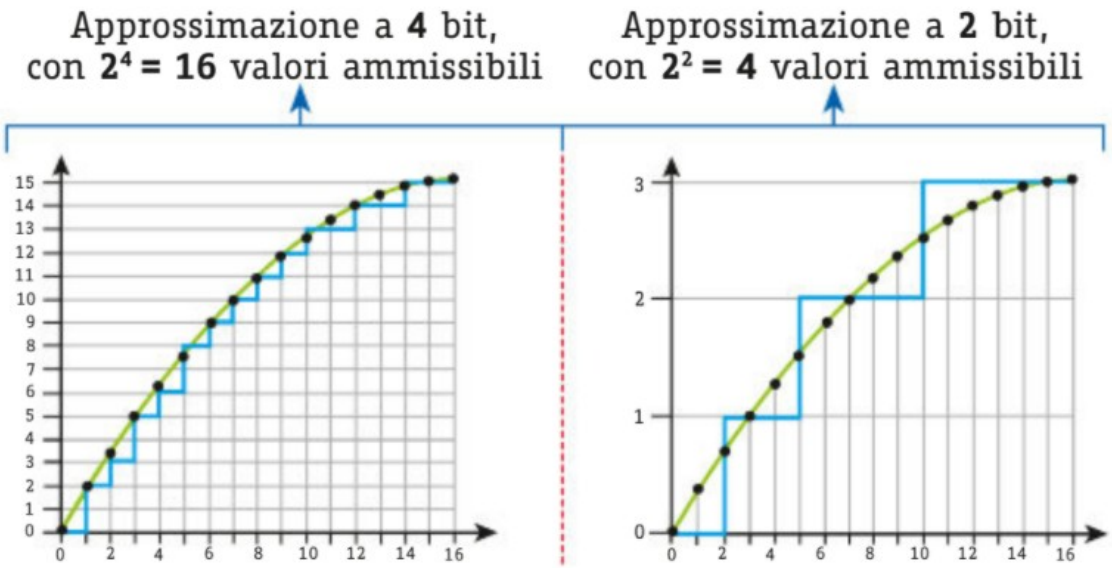


I software di gestione audio digitale come Audacity (che trattiamo nel seguito) mostrano delle forme d'onda che vengono campionate. Riducendo sempre più la scala delle X, dove è indicato il tempo, si arriva a vedere i punti dei singoli campionamenti. La distanza tra un punto e il successivo è di circa 0,0000227 secondi. Calcolando la frequenza di campionamento al secondo (1/0,0000227) si ottiene in questo caso 44100 hertz.



Il campionamento è costituito da una serio di misure, ciascuna chiamata campione, analogamente ai pixel di campionamento per le immagini. Ciascun campione è rappresentato nella memoria del computer da un valore numerico binario. La scelta del numero di bit necessari per rappresentare il segnale sonoro è molto importante per determinarne la precisione e definisce la profondità in bit, che determina l'intervallo dinamico (dynamic range) del segnale e rappresenta il rapporto fra i valori massimi e minimi registrati nell'intervallo di tempo.

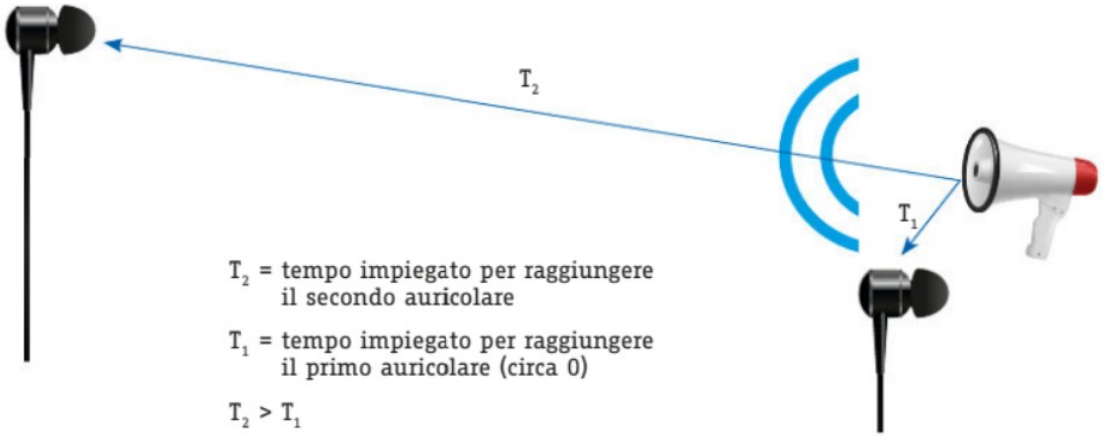
Esempio di campionamento a 2 e 4 bit: il grafico seguente mostra un campionamento effettuato a 4 bit nella prima metà (a sinistra), quindi a 2 bit nella seconda metà (a destra). I valori ammissibili per il secondo campionamento sono 4 (2^2) e vengono approssimati con una certa imprecisione. Nella prima metà, dove i bit di precisione sono 4. con 16 valori ammissibili (2^4). l’onda originale è approssimata decisamente meglio.


Approssimazione a 4 bit,
con 2⁴ = 16 valori ammissibili
Approssimazione a 2 bit,
con 2² = 4 valori ammissibili
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 2 4 6 8 10 12 14 16
0 1 2 3
0 2 4 6 8 10 12 14 16

Esercizio guidato: stimare la velocità del suono

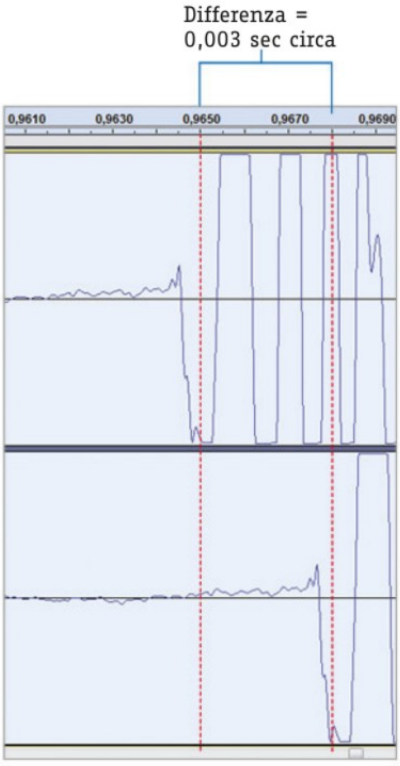
In questo esempio vogliamo mostrare come calcolare la velocita del suono a partire da una registrazione audio.

Per effettuare l’esperimento è necessario allontanare il più possibile i due auricolari delle cuffie tra di loro, rivolgendoli entrambi verso la stessa fonte sonora. Producendo un suono in prossimità di uno dei due auricolari, il secondo auricolare lo registrerà con un certo ritardo, a causa della distanza tra i due.



La velocità del suono potrà essere calcolata dato che sono noti i due dati principali: la distanza tra i due auricolari e il tempo impiegato dal suono per raggiungere il secondo altoparlante. La stima della velocità del suono risulterà migliore con l’aumentare della lunghezza del cavo che separa i due auricolari delle cuffie.

1.



2. In questo caso, avendo distanziato gli auricolari di circa 1 metro tra di loro e avendo ottenuto nel nostro esperimento il risultato visibile nella figura precedente (0,0029 secondi) possiamo affermare che la velocità del suono è:

$$V = \frac{S}{t} = \frac{1\,m}{0{,}0029\,s} = 343\,m/s$$

5

5. “Algoritmi”

Algoritmi

Un algoritmo è un procedimento che risolve un determinato problema attraverso un numero finito di passi elementari.

In informatica esso può essere rappresentato graficamente attraverso dei diagrammi di flusso, e formalizzato attraverso un linguaggio di programmazione[6] per scrivere un programma[7].

L'algoritmo è un concetto fondamentale dell'informatica, perché è alla base della nozione teorica di calcolabilità, ossia un problema si dice **calcolabile** quando è risolvibile mediante un algoritmo.

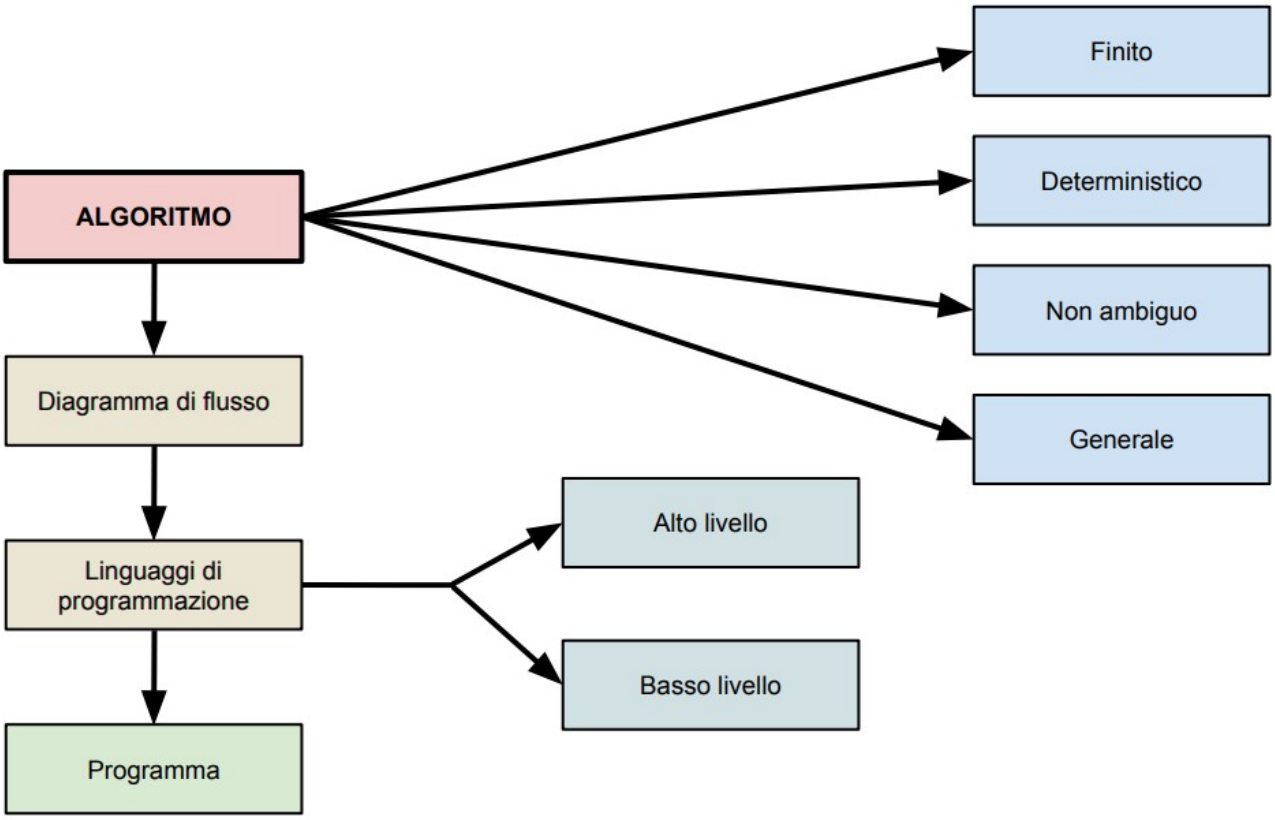
L'algoritmo è un concetto fondamentale anche nella fase di programmazione dello sviluppo di un software: preso un problema da automatizzare, la programmazione costituisce essenzialmente la traduzione di un algoritmo per tale problema in un programma, scritto in un certo linguaggio, che può essere quindi effettivamente eseguito da un computer.

Affinché un algoritmo sia funzionale alla logica della programmazione, deve avere le seguenti caratteristiche:

- **Finito**: L’algoritmo deve avere un numero finito di passaggi. Non può andare avanti all’infinito.
- **Non ambiguo**: L’algoritmo deve essere chiaro e non ambiguo. Ogni istruzione deve avere un solo significato e non può essere interpretata in modi diversi.
- **Deterministico**: L’algoritmo, dato un certo input, deve produrre sempre lo stesso output e segue sempre lo stesso insieme di passaggi.
- **Generale**: L’algoritmo deve essere applicabile a un insieme di problemi simili, non solo a un singolo problema specifico.
    - ***Esempio***: un problema specifico potrebbe essere “Determinare la misura dell’ipotenusa di un triangolo rettangolo i cui cateti misurano 5 cm e 7 cm”, ma sviluppare un algoritmo per questo tipo di problema vuol dire trovare una soluzione solo per i triangoli rettangoli che hanno i cateti delle dimensioni di 5 e 7 cm. Mentre invece un algoritmo ben progettato deve essere in grado di calcolare l’ipotenusa di un triangolo rettangolo per tutte le dimensioni possibili che possono avere i cateti, quindi

[6] È un linguaggio per la scrittura di programmi per computer. La maggior parte dei linguaggi di programmazione sono linguaggi formali basati su testo, ma possono anche essere di tipo grafico.
[7] Per programma (o applicazione) si intende la sequenza di istruzioni di un linguaggio di programmazione comprensibile al calcolatore che realizzano un determinato compito.

il problema dovrebbe essere riformulato come “Determinare la misura dell’ipotenusa di un triangolo rettangolo, essendo note le misure dei cateti.”



Per capire meglio, cos’è nella pratica un algoritmo, ecco alcuni esempi di algoritmi in linguaggio naturale:

Esempio di algoritmi (al di fuori dell’informatica):

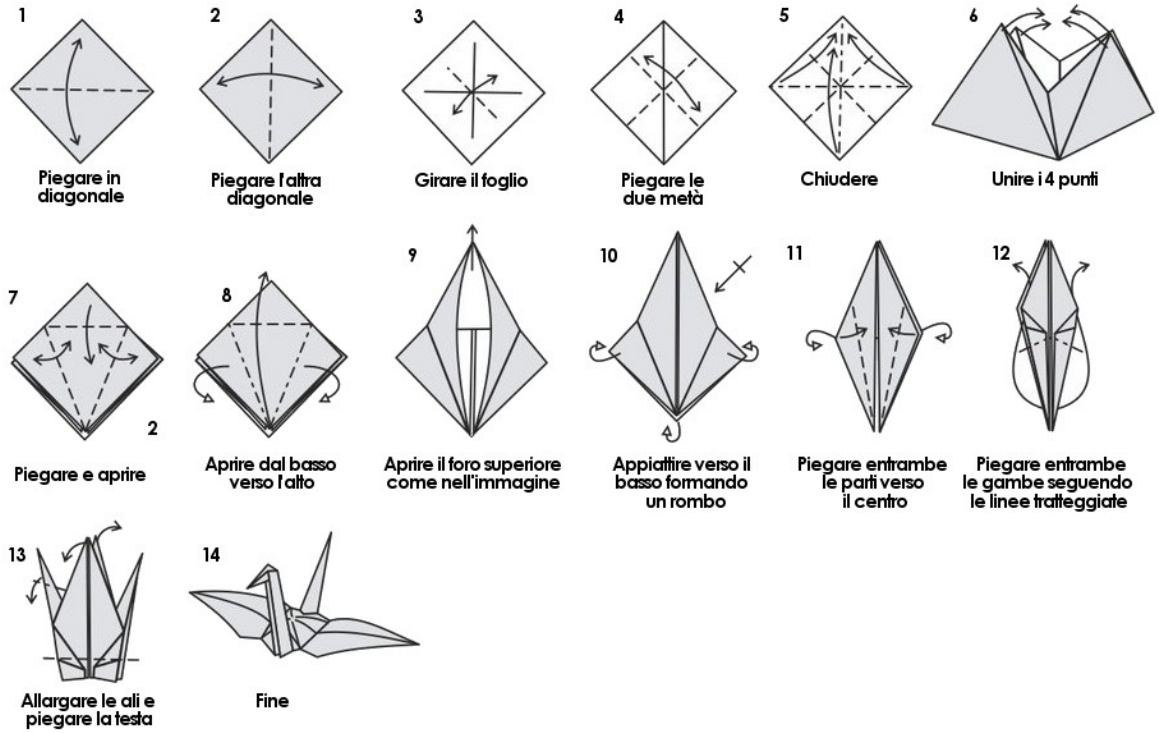
Procedimento che un professore utilizza per determinare chi è assente in una classe.

1. Prendere il registro della classe;
2. Per ogni alunno nella lista del registro, eseguire i seguenti passaggi:
    a. Chiamare l’alunno;
    b. Se l’alunno risponde “presente”, passare al prossimo alunno, altrimenti scrivere il nome dell’alunno sul registro nella sezione “assenti”;
3. Continuare questo processo fino a quando non si è chiamato ogni alunno nella lista del registro.

Procedimento per cuocere la pasta

1. Riempire una pentola con acqua e metterla sul fuoco;
2. Aggiungere un pizzico di sale all’acqua;
3. Quando l’acqua inizia a bollire, aggiungere la pasta;
4. Lasciare che la pasta cuocia per il tempo indicato sulla confezione;

5. Spegnere i fornelli e scolare la pasta.



Procedimento per fare la spesa

1. Preparare una lista della spesa scrivendo tutti gli articoli di cui si ha bisogno;
2. Prendere un carrello o un cestino all’ingresso del supermercato e iniziare a camminare lungo i corridoi del supermercato;
3. Per ogni articolo sulla lista, cercare l’articolo nel supermercato;
4. Quando si trova un articolo, lo si mette nel carrello o cestino;
5. Ripetere i 2 passaggi precedenti fino a quando non si trovano trovato tutti gli articoli della lista;
6. Una volta trovati tutti gli articoli, andare alla cassa a pagare;
7. Mettere tutti gli articoli nei sacchetti della spesa e portali a casa.

Esempi di algoritmi (informatici):

Scambio di valore di due variabili.

Supponiamo di avere due variabili, **a** e **b**, e vogliamo scambiare i loro valori. Ecco come potrebbe apparire l’algoritmo:

1. Creare una variabile **temporanea** e memorizzare il valore di **a**.
2. Assegnare il valore di **b** a **a**.
3. Assegnare il valore della variabile **temporanea** a **b**.

Trovare il numero più grande in una lista di numeri.
Supponiamo di avere una lista di numeri e vogliamo trovare il numero più grande. Ecco come potrebbe apparire l’algoritmo:

1. Impostare il primo numero della lista come il numero più grande finora.
2. Per ogni numero nella lista, confrontarlo con il numero più grande finora.
3. Se il numero corrente è più grande del numero più grande finora, allora diventa il nuovo numero più grande.
4. Continuare fino alla fine della lista.

Algoritmo “Bubble sort” (ordinamento a bolla), ordinamento di una lista.
Supponiamo di avere una lista di numeri e li si voglia ordinare in modo crescente. Ecco come potrebbe apparire l’algoritmo:

1. Confrontare il primo e il secondo numero nella lista.
2. Se il primo numero è maggiore del secondo, scambiarli.
3. Passare al prossimo paio di numeri (secondo e terzo) e ripetere il passaggio 2.
4. Continuare questo processo fino alla fine della lista.
5. Ripetere l’intero processo per la lunghezza della lista meno uno.

Rappresentazione di un algoritmo

In generale un algoritmo può essere rappresentato in:

- Linguaggio naturale
- Pseudocodice
- Diagramma di flusso
- Linguaggio di programmazione formale

Linguaggio naturale

Il termine "linguaggio naturale" si riferisce al linguaggio parlato o scritto che le persone utilizzano quotidianamente per comunicare tra loro. I linguaggi naturali comprendono lingue come l'italiano, l'inglese, il francese, il tedesco e tutte le lingue parlate dagli esseri umani. Questi linguaggi sono ricchi di ambiguità, contesto e varietà di significati, il che li rende comprensibili per gli esseri umani, ma spesso difficili da interpretare per le macchine.

Nel contesto della programmazione, l'opposto dei linguaggi naturali sono i "linguaggi di programmazione". I linguaggi di programmazione sono linguaggi formali creati appositamente per scrivere codice informatico. Questi linguaggi sono progettati per essere precisi, univoci e privi di ambiguità, in modo che le istruzioni possano essere eseguite da un computer senza errori di interpretazione.

Ecco alcune differenze chiave tra un linguaggio naturale e un linguaggio di programmazione:

1. **Ambiguità**: I linguaggi naturali possono essere ambigui e dipendono spesso dal contesto per comprendere appieno il significato delle parole o delle frasi. I linguaggi di programmazione sono privi di ambiguità e richiedono chiarezza e precisione.
2. **Variazione**: I linguaggi naturali variano ampiamente tra lingue, dialetti e registri. I linguaggi di programmazione sono standardizzati e seguono una sintassi specifica.
3. **Interpretazione**: I linguaggi naturali sono progettati per la comunicazione tra esseri umani. I linguaggi di programmazione sono progettati per essere compresi e eseguiti da un computer.
4. **Scopo**: I linguaggi naturali sono utilizzati per comunicare idee, pensieri e informazioni tra le persone. I linguaggi di programmazione sono utilizzati per scrivere istruzioni precise per un computer al fine di eseguire operazioni specifiche.

Per esempio, C++, Java, Python e altri linguaggi di programmazione sono esempi di linguaggi formali utilizzati per scrivere codice informatico, mentre l'italiano, l'inglese e altre lingue umane sono esempi di linguaggi naturali utilizzati per la comunicazione tra persone.

Si prenda in esempio l’algoritmo per lo scambio di valore tra due variabili descritto in precedenza, esso lo si può rappresentare in linguaggio naturale nel modo seguente:

1. Creare una variabile **temporanea** e memorizzare il valore di **a**.
2. Assegnare il valore di **b** a **a**.
3. Assegnare il valore della variabile **temporanea** a **b**.

Diagramma di flusso

Un diagramma di flusso (in inglese flow chart) è una rappresentazione grafica di un processo o di un algoritmo in programmazione. È una tecnica di visualizzazione utilizzata per rappresentare in modo chiaramente comprensibile il flusso di controllo all'interno di un programma o di una procedura. I

diagrammi di flusso sono ampiamente utilizzati in programmazione per scopi di progettazione, documentazione e comunicazione tra i membri del team di sviluppo.
Un diagramma di flusso aiuta i programmatori a visualizzare in modo chiaro la logica di un algoritmo o di un processo, facilitando la comprensione, la correzione degli errori e la comunicazione con altri membri del team. È uno strumento utile durante la fase di progettazione e può essere una risorsa preziosa per la documentazione del codice.
Tali diagrammi costituiscono lo strumento per presentare il ragionamento che condurrà al risultato atteso.

Esso consente di descrivere tramite un linguaggio di modellazione grafico:

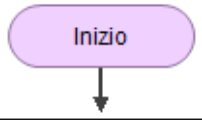
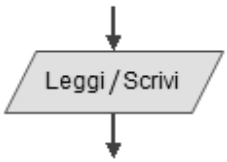
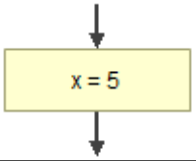
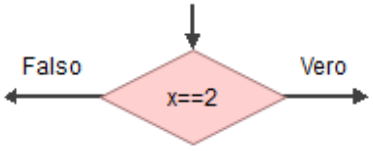
- le operazioni da compiere, rappresentate mediante sagome convenzionali (rettangoli, rombi, esagoni, parallelogrammi, rettangoli smussati...), ciascuna con un preciso significato logico e all'interno delle quali un'indicazione testuale descrive l'attività da svolgere;
- la sequenza nella quale devono essere compiute, rappresentata con frecce di collegamento.

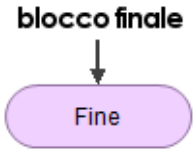
In algoritmi realizzati per la soluzione di problemi matematici, le istruzioni si sviluppano mediante costanti e variabili legate da operatore aritmetici (per esempio +, -, *, /, %), il cui risultato sarà assegnato a una variabile.

Ogni diagramma è caratterizzato da una lettura sequenziale:

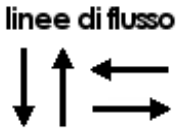
- si parte dal blocco iniziale
- si segue la freccia in uscita
- si giunge al blocco successivo e si effettua l'operazione descritta nel blocco
- si effettuano tutte le iterazioni fino a giungere al blocco finale.

Un diagramma di flusso è composto da vari simboli e frecce che rappresentano diversi elementi di un algoritmo, tra cui:

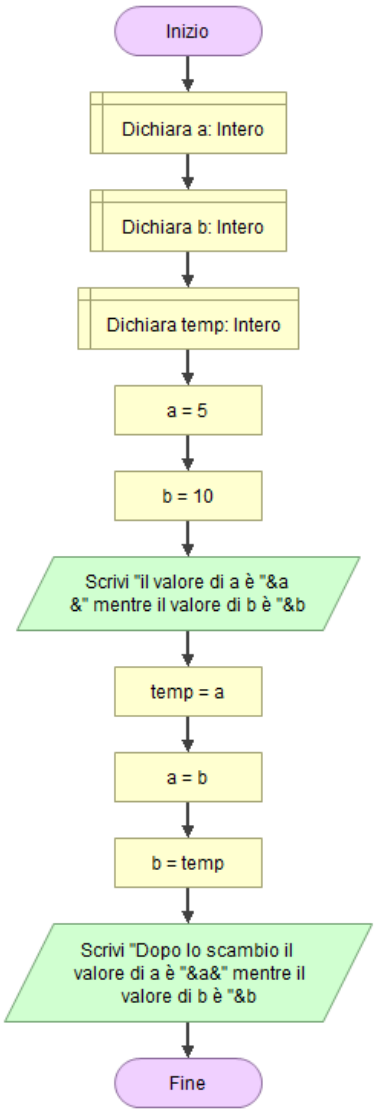
blocco iniziale 	Serve per indicare il punto di partenza dell'algoritmo e al suo interno possono apparire le scritte INIZIO o START. In un diagramma di flusso ne è presente solo uno.
blocco di input/output 	Viene utilizzato per le operazioni di ingresso o scrittura dei dati (input) e di uscita o lettura dei risultati (output). Ha una sola linea di giunzione in entrata e in uscita.
blocco di elaborazione 	Serve per le operazioni di calcolo o di assegnazione e al suo interno viene descritto in forma simbolica o letterale l'operazione da eseguire. Ha una sola linea di giunzione in entrata e in uscita.
blocco decisionale 	Serve per le operazioni di confronto, controllo o di scelta tra alternative per percorrere sequenze di istruzioni diverse in funzione del verificarsi o meno di eventi o situazioni. Al suo interno si inserisce l'espressione da considerare, il cui risultato viene confrontato con variabili e costanti attraverso gli operatori: • == (uguale a) • > (maggiore)

	• >= (maggiore uguale) • < (minore) • <= (minore uguale) • <> (diverso) Mentre il risultato è indicato sulle linee di giunzione: • V (vero) • F (falso)
	Serve per indicare la fine dell'algoritmo e al suo interno possono apparire le scritte FINE o STOP. In un diagramma di flusso ne è presente solo uno.

Inoltre:

	Collegano i vari simboli per mostrare il percorso sequenziale o condizionale che il programma segue.

Prendendo in esempio l’algoritmo per lo scambio di valore tra due variabili descritto in precedenza, esso lo si può rappresentare con un diagramma di flusso nel modo seguente:



Pseudocodice

Lo pseudocodice è un metodo di rappresentazione algoritmica che sta tra il linguaggio naturale (come l'italiano o l'inglese) e un linguaggio di programmazione formale (come Java, C++ o Python). Si tratta di un modo di scrivere algoritmi in una forma semi-strutturata, utilizzando una combinazione di linguaggio naturale e convenzioni di programmazione, al fine di descrivere un processo o un algoritmo in modo chiaro e comprensibile, senza la necessità di aderire a una sintassi di un linguaggio di programmazione specifico.

Lo pseudocodice è particolarmente utile nelle fasi iniziali di progettazione di un algoritmo o di un programma, in quanto consente di esprimere l'idea di base senza doversi preoccupare dei dettagli specifici di un linguaggio di programmazione. Questo lo rende uno strumento flessibile per la pianificazione e la comunicazione di idee algoritmiche.

Ovviamente non esiste uno pseudolinguaggio standard e convenzionalmente usato, in ogni caso ecco alcune caratteristiche comuni che dovrebbe avere lo pseudocodice:

1. Linguaggio semplificato: Lo pseudocodice utilizza un linguaggio semplificato, simile al linguaggio naturale, per descrivere passaggi e operazioni.

2. Sintassi informale: Non segue una sintassi rigorosa come un linguaggio di programmazione formale. Tuttavia, utilizza spesso convenzioni comuni per rappresentare concetti come cicli, decisioni e assegnazioni.
3. Focus sull'idea: Lo pseudocodice si concentra sull'espressione dell'idea o dell'algoritmo senza dettagli specifici del linguaggio.
4. Facilità di comprensione: È progettato per essere facilmente comprensibile da programmatori e non programmatori, facilitando la comunicazione delle logiche di programmazione.
5. Adattabilità: Può essere facilmente convertito in codice reale in un linguaggio di programmazione specifico quando si passa dalla fase di progettazione alla fase di implementazione.

Prendendo in esempio l’algoritmo per lo scambio di valore tra due variabili descritto in precedenza, esso lo si può rappresentare in pseudocodice nel modo seguente:

```
Inizio
    Dichiarare a, b, temp come interi
    a = 5
    b = 10
    Stampa "Prima dello scambio: a =", a, "b =", b
    temp = a
    a = b
    b = temp
    Stampa "Dopo lo scambio: a =", a, "b =", b
Fine
```

Linguaggi di programmazione

Un linguaggio di programmazione è un sistema formale progettato per comunicare istruzioni a un computer. Essi forniscono un insieme di regole e convenzioni che consentono ai programmatori di definire algoritmi e istruzioni in un formato comprensibile sia per gli esseri umani che per le macchine.
I linguaggi di programmazione svolgono un ruolo fondamentale nello sviluppo di software e nella comunicazione tra i programmatori e i computer.
Il codice scritto in un determinato linguaggio di programmazione prende il nome di **codice sorgente.**

Un linguaggio di programmazione ha delle:

- **Componenti chiave:**
  - **Sintassi**: Regole che definiscono come le istruzioni devono essere scritte.
  - **Semantica**: Il significato delle istruzioni e come vengono interpretate dal compilatore o interprete.
- **Funzionalità tipiche:**
  - Variabili e costanti
  - Operazioni aritmetiche, relazionali e logiche
  - Strutture di controllo (condizionali, cicli, ecc.)
  - Definizione e chiamata di funzioni
  - Gestione dell'input/output

Classificazione dei linguaggi di programmazione

Un linguaggio di programmazione può essere classificato:

1. In base al livello di astrazione:

- **Linguaggi di basso livello:** Più vicini al linguaggio macchina, offrono poco o nessun livello di astrazione. Hanno come caratteristiche il controllo diretto dell'hardware e un codice specifico per l'architettura.
  - **Vantaggi**: Massima efficienza e controllo.
  - **Svantaggi**: Difficili da scrivere e mantenere, non portabili.
  - **Esempi:**
    - **Assembly x86:** Utilizzato per la programmazione di basso livello su processori x86.
    - **Assembly ARM:** Comune nella programmazione di dispositivi embedded e mobile.
- **Linguaggi di medio livello:** Forniscono alcune astrazioni ma mantengono un certo controllo sull'hardware. La loro caratteristica è che combinano elementi di alto e basso livello.
  - **Vantaggi**: Buon equilibrio tra controllo e astrazione.
  - **Svantaggi**: Possono richiedere una comprensione più profonda dell'architettura sottostante.
  - **Esempi:**
    - **C:** Ampiamente utilizzato per lo sviluppo di sistemi operativi e software di sistema.
- **Linguaggi di alto livello:** Offrono forti astrazioni e sono più vicini al linguaggio umano. Hanno come caratteristica l’indipendenza dall'hardware.
  - **Vantaggi**: Più facili da imparare e usare, maggiore produttività.
  - **Svantaggi**: Meno controllo diretto sull'hardware, potenzialmente meno efficienti.

    - **Esempi**:
        - C++: Combina astrazione e controllo; offre funzionalità orientate agli oggetti mantenendo efficienza simile al C.
        - Java: Orientato agli oggetti; fornisce portabilità attraverso la JVM e gestione automatica della memoria per maggiore produttività.
- **Linguaggi di altissimo livello:** Offrono un livello di astrazione molto elevato, avvicinandosi al linguaggio naturale o a notazioni di dominio specifico. Hanno come caratteristiche una sintassi molto vicina al linguaggio umano e sono spesso orientati a risolvere problemi in domini specifici.
    - **Vantaggi:** Facilità di apprendimento e uso, rapida prototipazione e sviluppo di applicazioni e riduzione degli errori di programmazione
    - **Svantaggi:** Possono essere meno efficienti in termini di prestazioni, hanno un controllo limitato sui dettagli di basso livello e potrebbero non essere adatti per tutti i tipi di applicazioni
    - **Esempi**:
        - **Scratch**: Linguaggio di programmazione visuale per l'educazione. Utilizza blocchi grafici per costruire programmi, rendendo la programmazione accessibile ai principianti
        - **Python**: Noto per la sua semplicità e leggibilità, ampiamente usato in data science e web development.
        - **SQL (Structured Query Language):** Utilizzato per la gestione e l'interrogazione di database relazionali. Permette di esprimere operazioni complesse sui dati in modo dichiarativo
        - **MATLAB**: Focalizzato sul calcolo numerico e l'elaborazione di segnali. Fornisce un ambiente integrato per calcoli matematici, visualizzazione e programmazione

2. In base al tipo di paradigma:

- **Imperativo**: L'approccio imperativo si concentra sull'esecuzione sequenziale delle istruzioni e sulla modifica dello stato delle variabili per ottenere il risultato desiderato. Gli algoritmi vengono espressi come una serie di comandi che modificano le variabili e il flusso di esecuzione. (esempi: C, Pascal)
- **Orientato agli oggetti:** È un tipo di programmazione basata su oggetti e classi, si basa sul concetto di "oggetti", che sono istanze di classi. Gli oggetti contengono dati e metodi che operano su tali dati. (esempi: Java, C++, Python)
- **Dichiarativo**: Il paradigma dichiarativo si concentra su cosa dovrebbe essere fatto, senza specificare come farlo. Gli algoritmi vengono definiti in modo dichiarativo, fornendo le regole o le condizioni che guidano l'esecuzione senza dettagli implementativi. (esempi: SQL)
- **Orientata agli eventi:** È un paradigma di programmazione che si basa sulla gestione di eventi e risposte agli eventi. In questo paradigma, il flusso del programma è guidato dagli eventi che si verificano, come clic del mouse, pressioni di tasti, input utente o notifiche di sistema. (esempi: Visual Basic, JavaScript)

3. In base all’ambiente di esecuzione:

- **Compilati**: Il codice sorgente viene tradotto in linguaggio macchina prima dell'esecuzione. In particolare i linguaggi compilati hanno le seguenti caratteristiche:
    - **Fase di compilazione**: In un linguaggio compilato, il codice sorgente viene tradotto in linguaggio macchina in una fase separata chiamata "compilazione". Il risultato di

questa compilazione è un file eseguibile specifico per la piattaforma su cui viene eseguito.
    - **Esecuzione:** Dopo la compilazione, il programma eseguibile può essere eseguito direttamente senza ulteriori passaggi di traduzione. Il sistema operativo interpreta direttamente il codice macchina contenuto nell'eseguibile.
    - **Prestazioni**: I programmi compilati tendono ad avere prestazioni più elevate rispetto ai programmi interpretati, poiché non è necessario tradurre il codice durante l'esecuzione. L'eseguibile compilato può sfruttare appieno le ottimizzazioni del compilatore.
- **Interpretati**: Il codice sorgente viene eseguito riga per riga da un interprete. In particolare i linguaggi interpretati hanno le seguenti caratteristiche:
    - **Fase di Interpretazione:** In un linguaggio interpretato, il codice sorgente non viene tradotto in linguaggio macchina durante la fase di compilazione. Invece, viene eseguito da un interprete linea per linea o istruzione per istruzione durante l'esecuzione del programma.
    - **Esecuzione**: L'interprete legge ed esegue il codice sorgente direttamente. Ogni istruzione viene tradotta in codice macchina contemporaneamente al programma in esecuzione.
    - **Prestazioni**: I programmi interpretati tendono ad avere prestazioni inferiori rispetto ai programmi compilati poiché c'è un consumo maggiore di risorse associato all'interpretazione istruzione per istruzione durante l'esecuzione.

4. In base all’approccio alla memoria:

- **Gestione manuale della memoria**: Il programmatore è responsabile della gestione della memoria. (esempio: C o C++)
- **Gestione automatica della memoria**: Il linguaggio si occupa automaticamente dell'allocazione e della deallocazione della memoria (esempi: Java, Python).

Prendendo in esempio l’algoritmo per lo scambio di valore tra due variabili, esso lo si può implementare in vari linguaggi di programmazione nei modi seguenti:

Esempio in Java

```
public class Main {
    public static void main(String[] args) {
        int a = 5;
        int b = 10;
        System.out.println("Prima dello scambio: a = "+a+", b = "+b);
        int temp = a;
        a = b;
        b = temp;
        System.out.println("Dopo lo scambio: a = "+a+", b = "+b);
    }
}
```

Esempio in C++

```
#include<iostream>
using namespace std;
int main() {
    int a = 5;
    int b = 10;
    cout << "Prima dello scambio: a = " << a << ", b = " << b << endl;
    int temp = a;
    a = b;
    b = temp;
    cout << "Dopo lo scambio: a = " << a << ", b = " << b << endl;
    return 0;
}
```

Esempio in Python

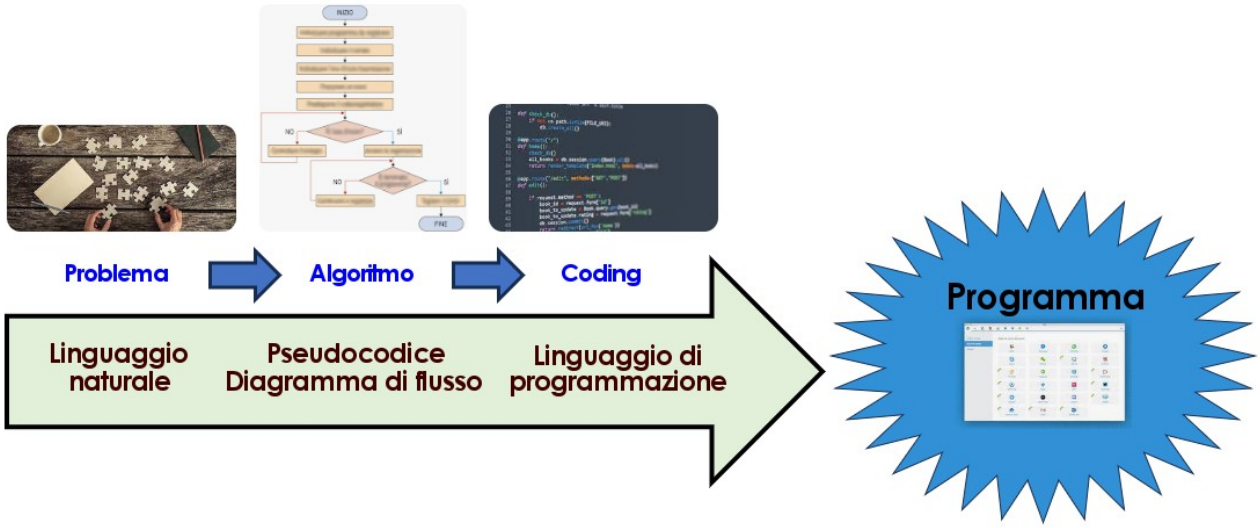
```
a = 5
b = 10
print(f"Prima dello scambio: a = {a}, b = {b}")
temp = a
a = b
b = temp
print(f"Dopo lo scambio: a = {a}, b = {b}")
```

Programma

Il termine "programma" si riferisce a un insieme di istruzioni scritte in un linguaggio di programmazione che definiscono una serie di operazioni da eseguire da parte di un computer o di un sistema informatico. Un programma è fondamentalmente un elenco di istruzioni dettagliate che indicano al computer cosa fare, passo dopo passo, al fine di svolgere una specifica attività o compito.

Un programma può essere molto semplice o estremamente complesso, a seconda delle necessità dell'applicazione. Ad esempio, un programma può essere un semplice script che calcola la somma di due numeri, o può essere un sofisticato sistema operativo che gestisce tutte le funzioni di un computer.

Con un computer indichiamo sia lo strumento per la rappresentazione e l'elaborazione dell'informazione sia un esecutore di algoritmi.



I programmi possono interagire con l'ambiente esterno attraverso l'input e l'output. L'input può provenire da tastiere, mouse, sensori o altri dispositivi, mentre l'output può essere visualizzato su schermi, stampato su carta o inviato tramite reti.

Differenza tra algoritmo e programma

Il termine programma indica la scrittura di un algoritmo per uno specifico compito, mentre un algoritmo è definito come la descrizione della soluzione di un problema espresso tramite operazioni che l'esecutore dell'algoritmo capisce e può eseguire.

Programmare dipende dal compito da eseguire mentre un algoritmo trasmette la descrizione di una soluzione ad un determinato problema.

Programmazione

Per programmazione si intende l'attività con cui definiscono le operazioni che servono a predisporre l'elaboratore ad eseguire un particolare insieme di azioni su particolari dati, allo scopo di risolvere un problema.

6. “Diagrammi di flusso con Flowgorithm”

Introduzione

Flowgorithm è uno strumento di programmazione di tipo grafico che consente agli utenti di scrivere ed eseguire programmi utilizzando i diagrammi di flusso. Il diagramma di flusso può essere convertito in diversi linguaggi di programmazione.

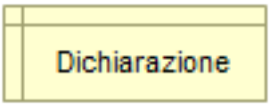
Creazione di una variabile

Una variabile è un’entità in cui viene memorizzato un valore. Le variabili vengono utilizzate per memorizzare dati che devono essere utilizzati dal programma.

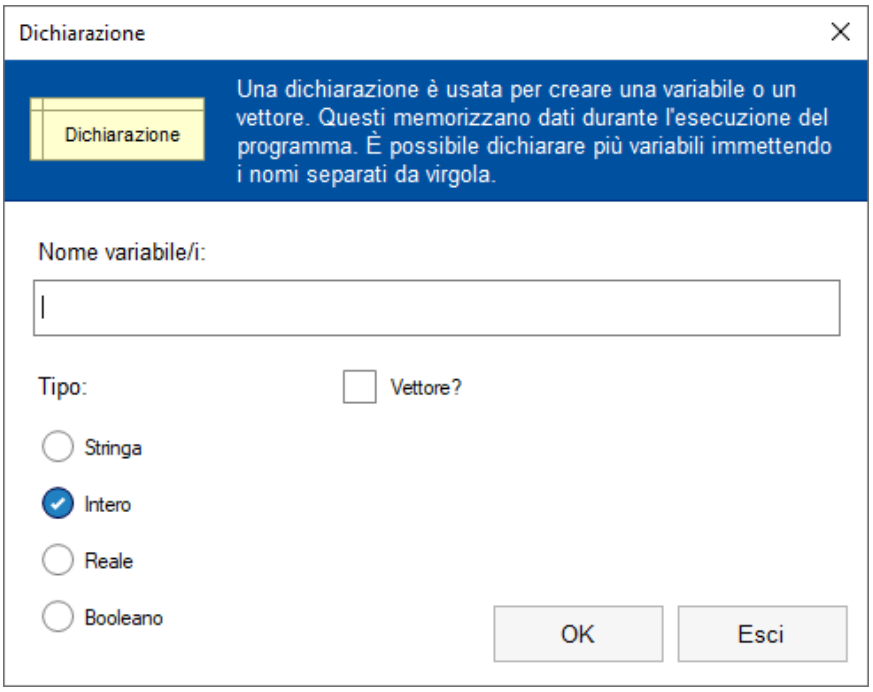
Una variabile è costituita da tre parti:

- **Nome**: identifica la variabile. Deve essere un identificatore valido per il linguaggio di programmazione in uso.
- **Tipo**: indica il tipo di dato che può essere memorizzato nella variabile. I tipi di dati possono essere numerici, di stringhe o booleani.
- **Valore**: il valore memorizzato nella variabile. Il valore di una variabile può essere modificato durante l'esecuzione del programma

La creazione di una variabile avviene tramite la sua **dichiarazione**, ed è rappresentata dal seguente blocco:



Il blocco **“Dichiarazione”** nella sua compilazione si presenta in questo modo:



I nomi delle variabili (chiamati identificatori) devono seguire le seguenti regole:

- Devono iniziare con una lettera.
- Dopo la prima lettera, l’identificatore può contenere lettere o numeri.
- Non sono permessi gli spazi.

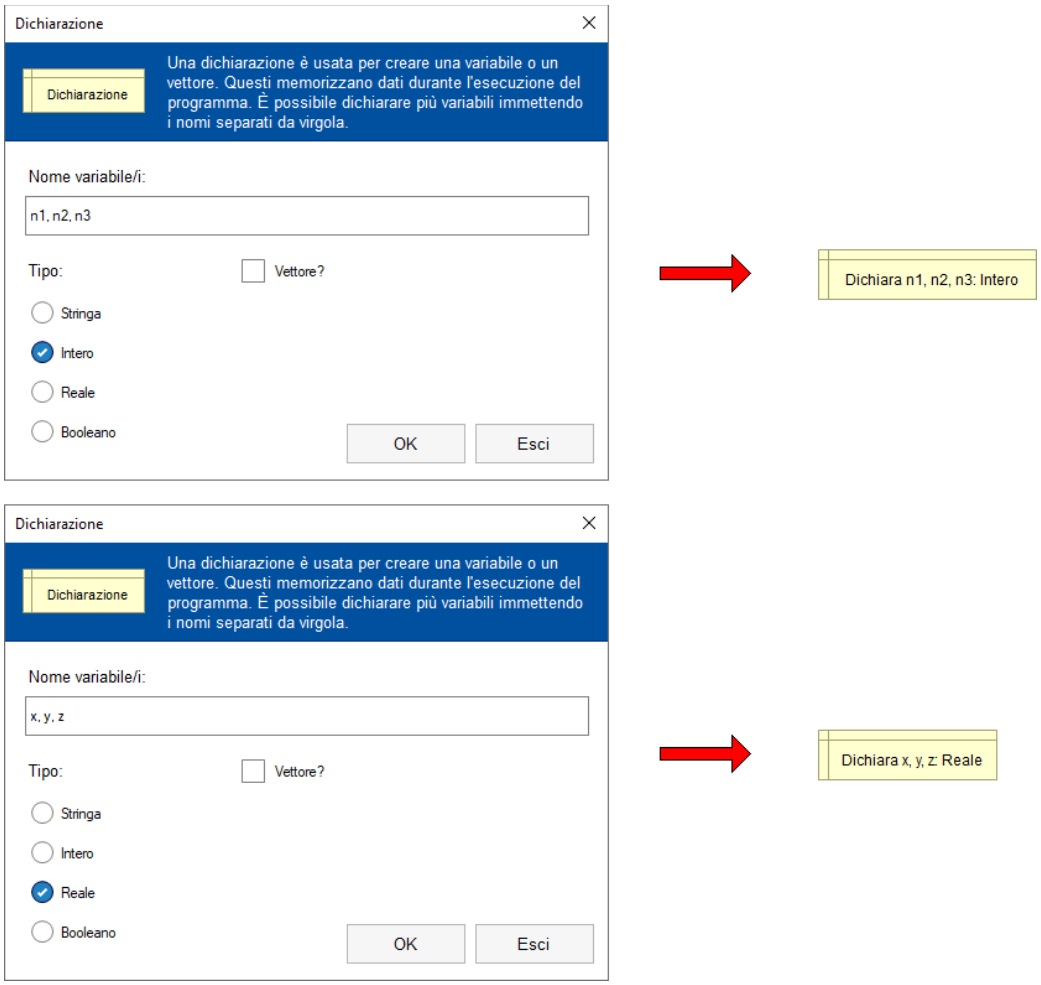
Una variabile deve avere un nome, è può contenere valori di quattro tipi:

- Stringa (testo);

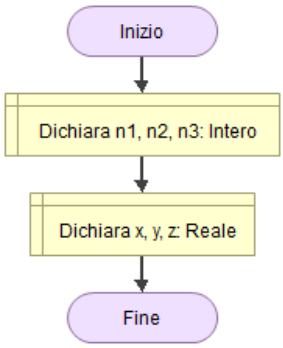
- Intero (numeri interi);
- Reale (numeri con la virgola);
- Booleano (vero o falso, o più precisamente **true** e **false**).

Il blocco dichiarazione serve anche per creare dei vettori (che verranno illustrati in seguito).

È possibile dichiarare più variabili dello stesso tipo in uno stesso blocco, immettendo i nomi separati da una virgola.



Se si ha la necessita di creare più variabili di tipo diverso, bisogna creare un blocco per ogni tipo di variabile, come nell’esempio:



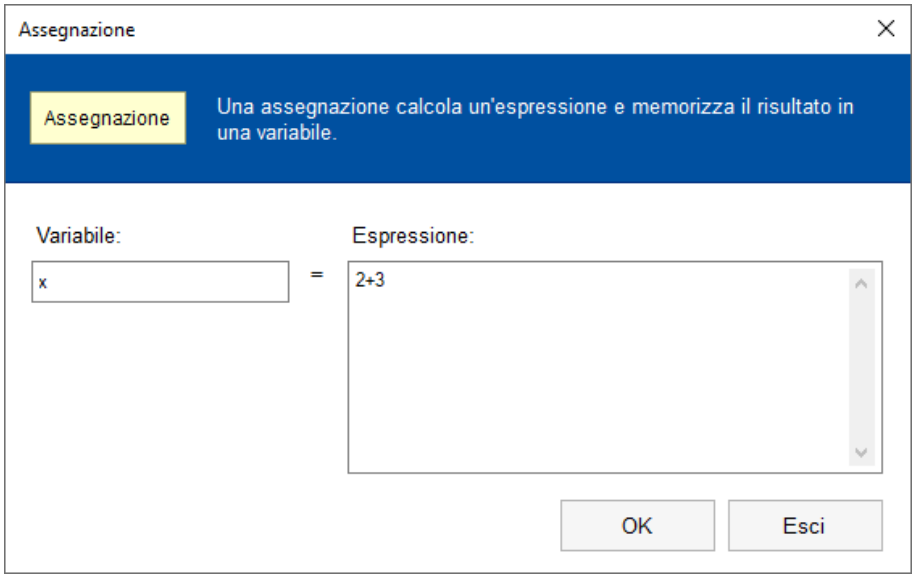
Assegnare un valore ad una variabile

Per assegnare un valore ad una variabile si possono usare due blocchi, quello di **lettura** o di **assegnazione**:

Nel blocco di **“Lettura”** (o inserimento da tastiera), il valore alla variabile verrà assegnato durante l’esecuzione del programma, e quindi in un certo senso il suo valore sarà ignoto durante la costruzione del diagramma di flusso.

Nel blocco di **“Assegnazione”**, il valore alla variabile verrà definito durante la creazione del diagramma di flusso e può contenere **valori** o **espressioni matematiche**.

Il blocco **“Assegnazione”** nella sua compilazione si presenta in questo modo:



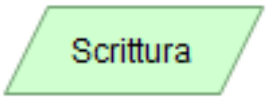
Operatori aritmetici

Gli operatori aritmetici usati per definire espressioni o eseguire calcoli sono i seguenti:

- **+** addizione
- **-** sottrazione
- ***** moltiplicazione
- **/** divisione
- **%** modulo (restituisce il resto di una divisione)

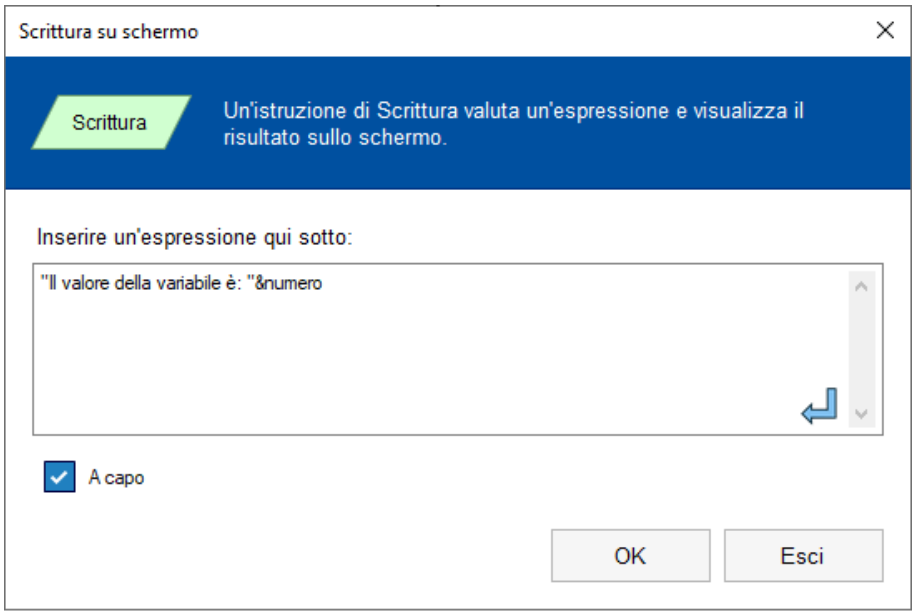
Stampare a video un testo e/o il valore di una variabile

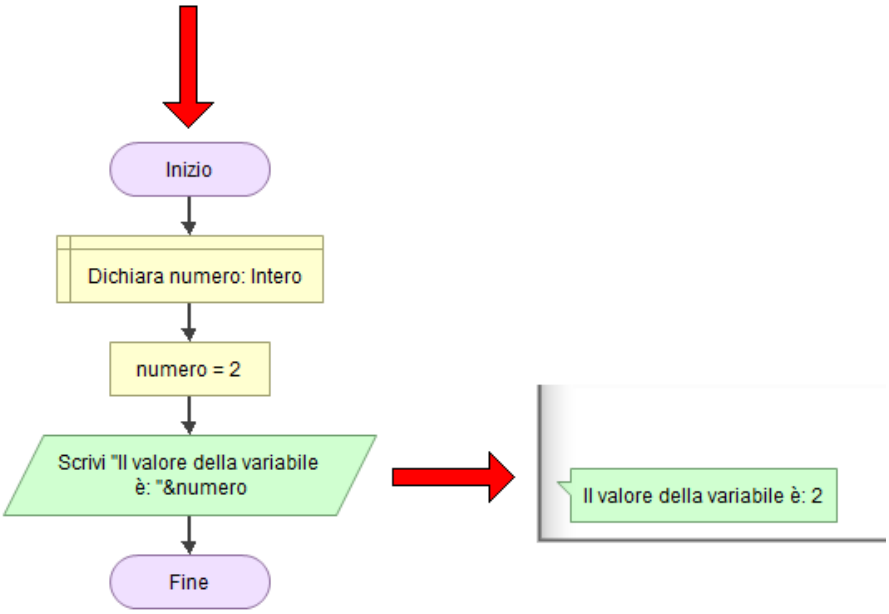
Per stampare a video un testo o una variabile è necessario utilizzare il blocco **“Scrittura”**:



All’interno del blocco scrittura tutto il **testo** che si vuole stampare a video va messo **“tra virgolette”** (chiamate doppi apici) in modo che venga riconosciuto come stringa (ossia come un testo), mentre se si vuole stampare a video il valore di una **variabile**, va inserito il nome della variabile **senza le virgolette**.

Se invece all’interno del blocco di “Scrittura” si vuole inserire sia un testo sia il valore di una variabile, è possibile utilizzare l’operatore di concatenazione **&**.





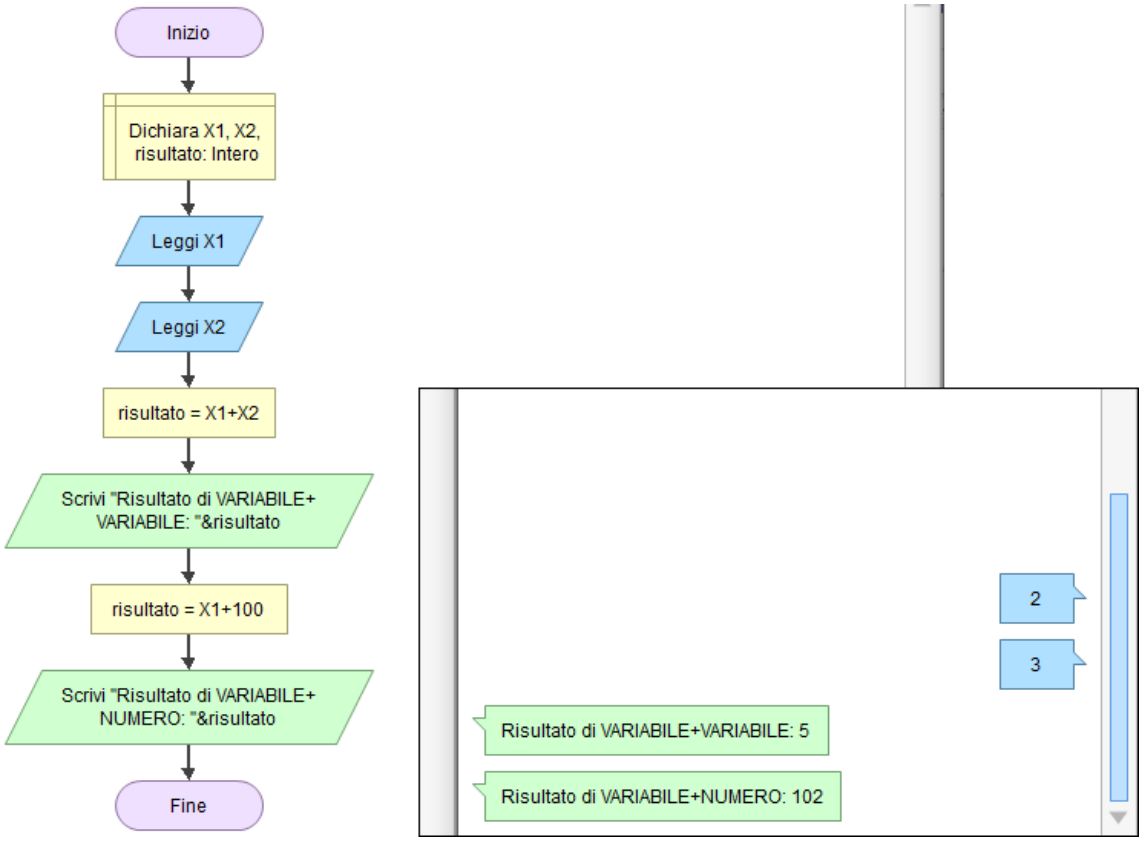
Unire più variabili

L’unione tra variabili e formati compatibili è detta “concatenazione”, e bisogna fare la distinzione tra variabili numeriche (come interi e/o reali che sono compatibili tra di loro) e variabili di tipo stringa (dove anche i numeri sono trattati come caratteri).

Unione tra variabili di tipo numerico

Se si vuole effettuare una somma tra variabili di tipo numeriche (intero e/o reali) si deve usare l’operatore “+” (addizione).
Quindi nel blocco di “assegnazione” si possono trovare queste 2 casistiche:

- variabile = variabile+NUMERO
- variabile = variabile1+variabile2

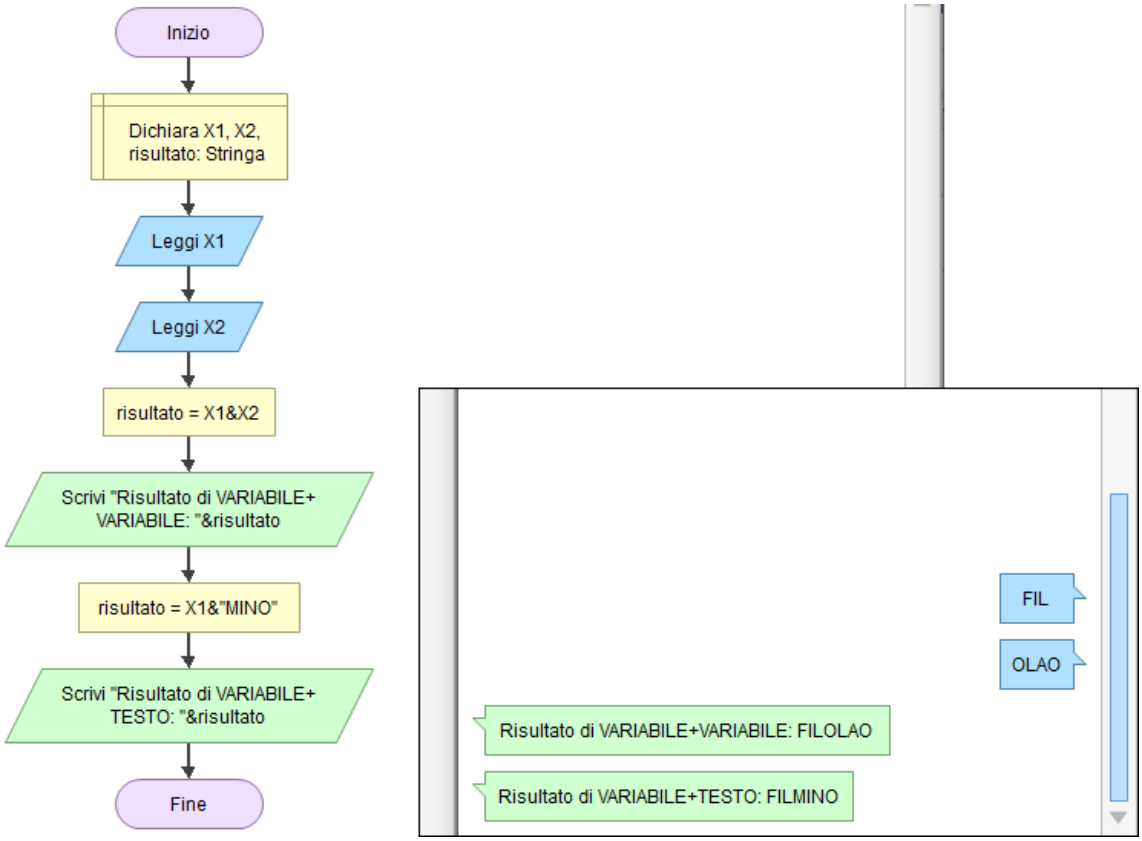


Ovviamente tra le variabili di tipo numerico è possibile effettuare tutte le operazioni aritmetiche (addizione, sottrazione, moltiplicazione, divisione e resto) usando gli operatori aritmetici visti in precedenza (+ - * / %).

Unione tra variabili di tipo stringa

Se si vuole effettuare una unione tra variabili di tipo stringa si deve usare l’operatore “&” (concatena).

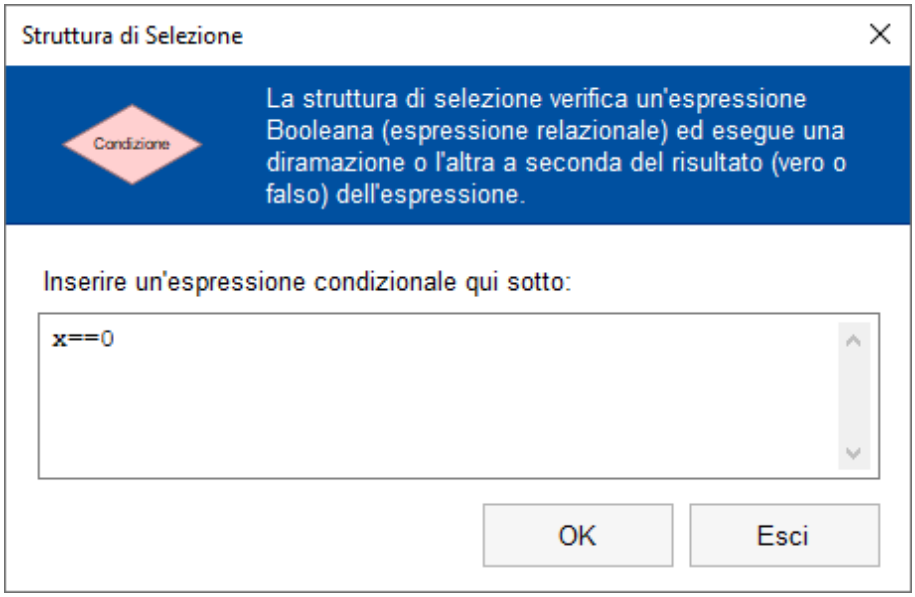
- variabile = variabile&”TESTO”
- variabile = variabile1&variabile2



Condizione

Il blocco “Condizione” controlla una espressione booleana e passa al ramo del Vero o del Falso a seconda se è verificata o meno la condizione.

Nel blocco di **“Condizione”** va inserito un confronto tra due variabili (o tra una variabile ed un valore, o anche tra i risultati di espressioni), se questo confronto viene soddisfatto allora il flusso seguirà la diramazione **“Vero”**, altrimenti seguirà la diramazione **“Falso”.**



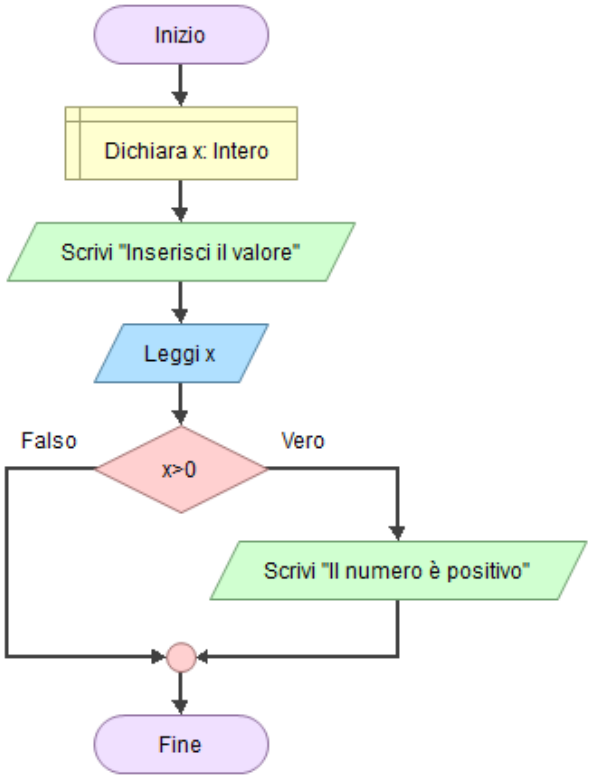
Operatori relazionali

Per effettuare un confronto si fa uso degli operatori relazionali, che sono i seguenti:

>	maggiore di
>=	maggiore uguale di
<	minore di
<=	minore uguale di
==	uguale a
!=	non uguale a

Ad esempio se si vuole verificare che il valore di una variabile sia positivo o meno è possibile schematizzarlo nel modo seguente:

Nota: Non è detto, che la ramificazione del lato “Falso” debba necessariamente seguire l’esecuzione di qualche blocco:

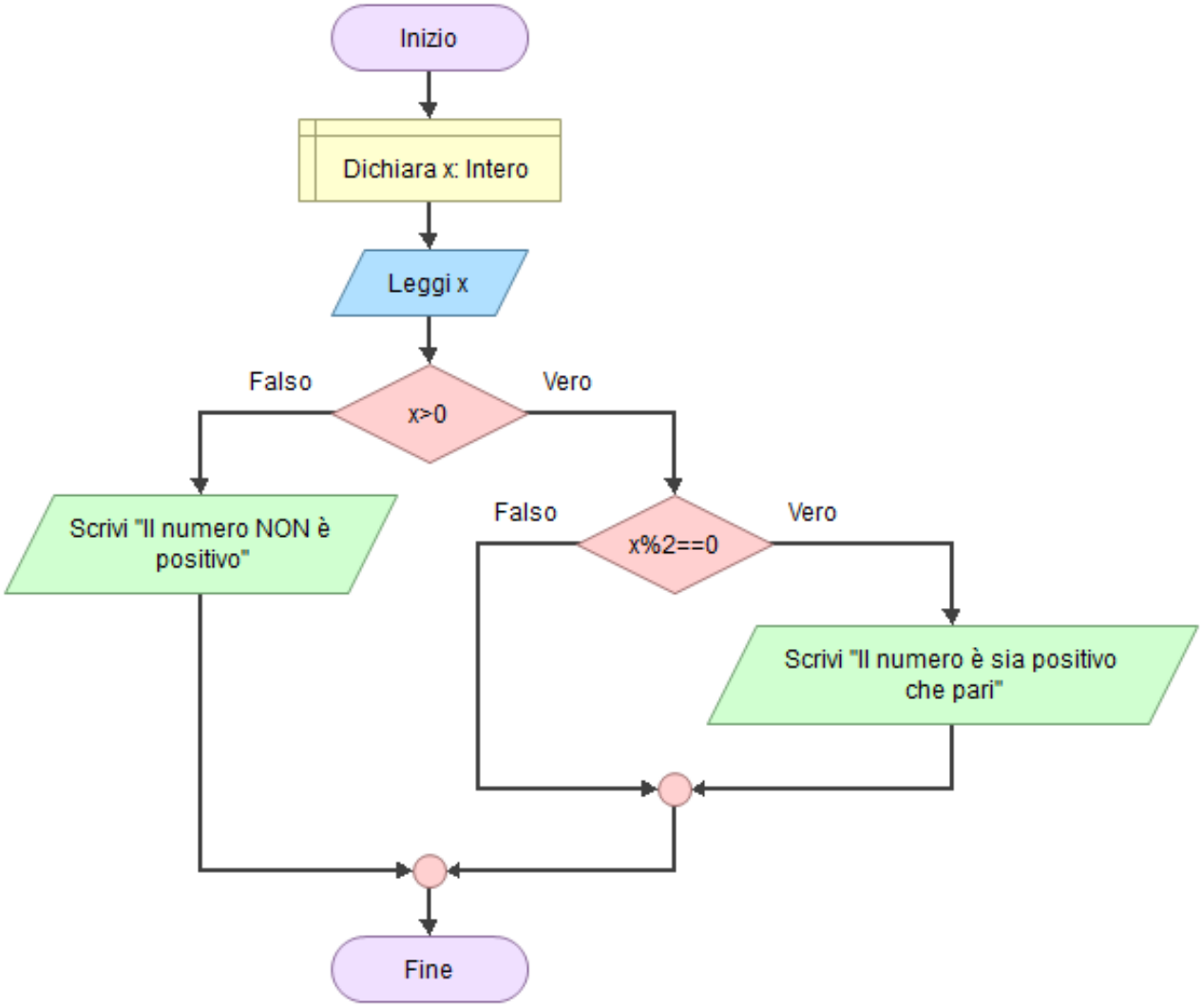


Nel caso si voglia verificare più di una condizione per arrivare alla risoluzione di un algoritmo, si possono seguire due strade:

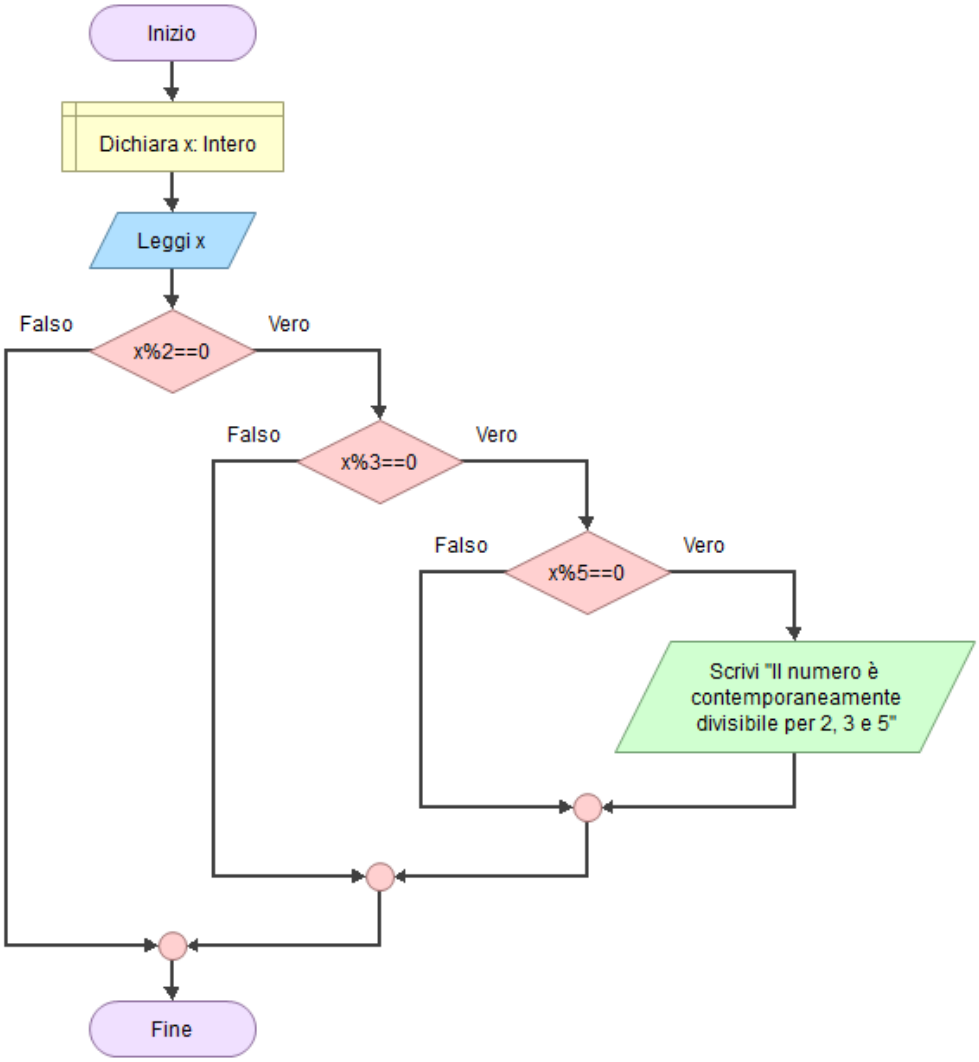
- ramificare le condizioni
- inserire, se possibile, tutte le condizioni all’interno di un unico blocco, attraverso gli operatori logici.

Condizioni ramificate

Ad esempio si supponga di voler verificare che un numero intero X in ingresso sia contemporaneamente sia positivo che pari:


Inizio
Dichiara x: Intero
Leggi x
Falso
x>0
Vero
Scrivi "Il numero NON è positivo"
Falso
x%2==0
Vero
Scrivi "Il numero è sia positivo che pari"
Fine

Altro esempio verificare che un numero intero X sia contemporaneamente divisibile per 2, per 3 e per 5:



Operatori logici

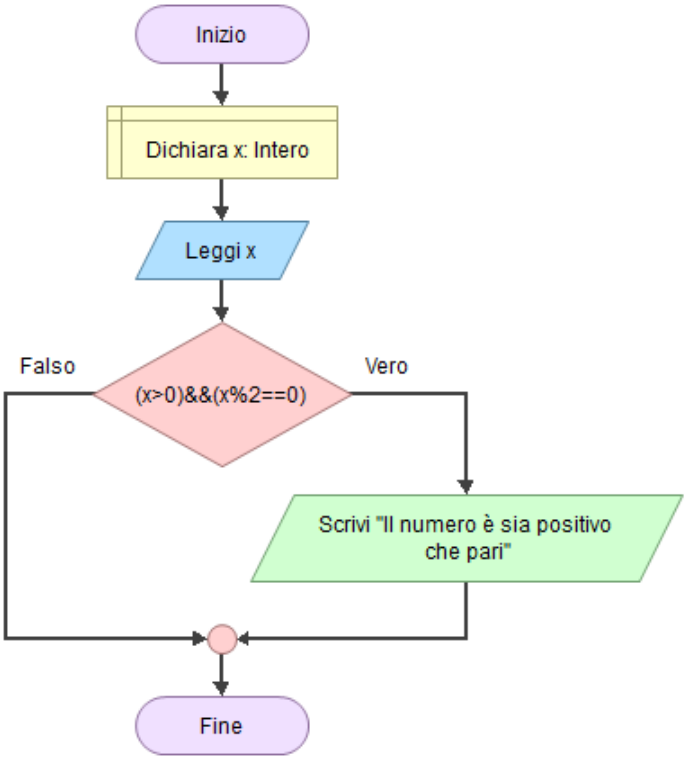
È possibile evitare la ramificazione dei costrutti condizionali attraverso gli operatori logici.

- **&&,** (che significa AND), ossia tutte e 2 condizioni devono essere soddisfatte;
- || (che significa OR), ossia 1 sola condizione deve essere soddisfatta.

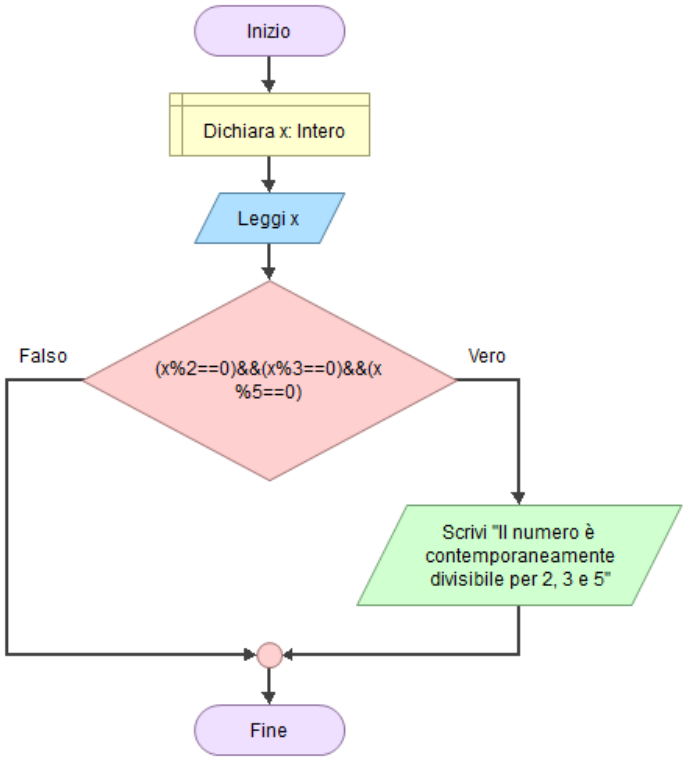
In pratica, in un blocco condizione, è possibile inserire come espressione condizionale (condizione1)&&(condizione2)&&(condizione3), che vuol dire che tutte e tre le condizioni devono essere verificate affinché si possa passare al ramo “Vero” della condizione.

In (condizione1), (condizione2) e (condizione3) devono essere inserite delle espressioni di confronto come ad esempio x>0 oppure x%2==0.

Esempio, prendendo in considerazione l’esempio precedente, ossia, si supponga di voler verificare che un numero intero X in ingresso sia contemporaneamente sia positivo che pari, questo diventa:



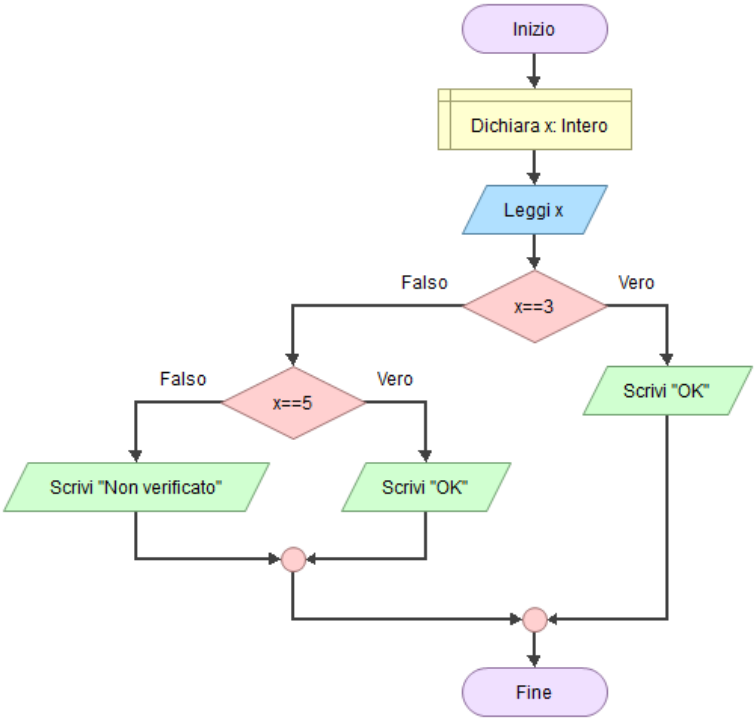
Invece, prendendo in considerazione l’esempio di verificare che un numero intero X sia contemporaneamente divisibile per 2, per 3 e per 5, esso diventa:



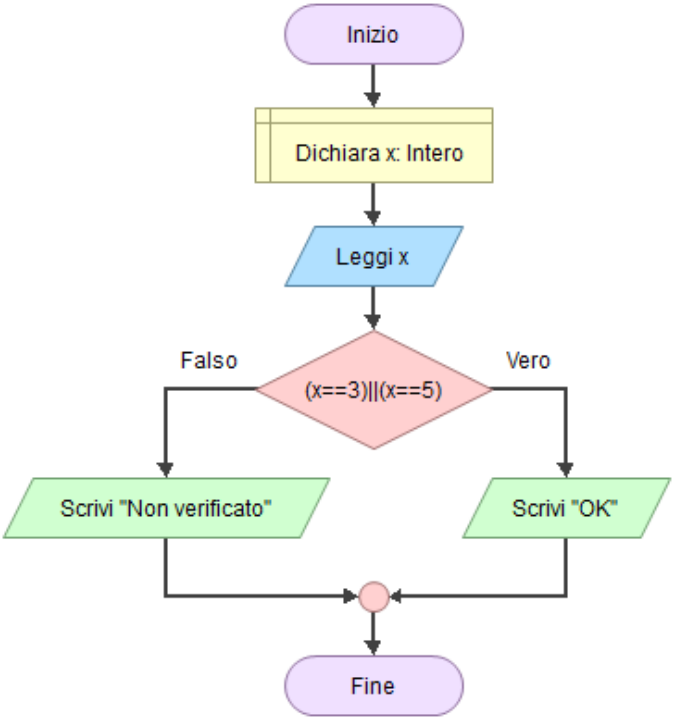
Per capire invece come usare l’operatore logico “OR” (che si indica con la doppia barra verticale “||” che è possibile scrivere con la combinazione di tasti SHIFT + “\”, di fianco il tasto “1”) ci si può aiutare con un esempio.

Dato un numero intero X in ingresso, verificare se è uguale a 3 oppure se è uguale a 5 scrivendo in output “OK”, mentre se è diverso stampare in output “Non verificato”.

Si può usare una nidificazione delle condizioni come in figura:



Oppure semplicemente:



7. “Fondamenti di programmazione in Python”

Introduzione al linguaggio Python

Python è un linguaggio di programmazione ad alto livello, interpretato e orientato agli oggetti, sviluppato originariamente da Guido van Rossum nel 1991. È uno dei linguaggi di programmazione più popolari al mondo, utilizzato per creare una vasta gamma di applicazioni.

Python è noto per la sua sintassi chiara e leggibile, che enfatizza la leggibilità del codice e riduce il costo di manutenzione del software. È un linguaggio multiparadigma che supporta la programmazione orientata agli oggetti, imperativa e funzionale.

Questo linguaggio è ampiamente utilizzato in molti campi, tra cui lo sviluppo web, l'analisi dei dati, l'intelligenza artificiale, l'apprendimento automatico, lo sviluppo di script di sistema e molti altri ambiti dove la facilità d'uso e la produttività sono cruciali.

Caratteristiche di Python

Python ha una serie di caratteristiche che lo rendono un linguaggio di programmazione versatile e potente:

- **Semplicità e Leggibilità:** La sintassi di Python è progettata per essere chiara e intuitiva, rendendo il codice facile da leggere e scrivere.
- **Programmazione Orientata agli Oggetti (OOP):** Python supporta pienamente i concetti di OOP, inclusi incapsulamento, ereditarietà e polimorfismo, permettendo agli sviluppatori di creare programmi modulari e riutilizzabili.
- **Gestione Automatica della Memoria:** Python utilizza un *garbage collector*, che gestisce automaticamente la memoria, liberando gli sviluppatori da questo compito e riducendo i rischi di errori di gestione della memoria.
- **Tipizzazione Dinamica:** Python è un linguaggio a tipizzazione dinamica, il che significa che non è necessario dichiarare il tipo di una variabile prima di usarla.
- **Vasta Libreria Standard:** Python viene fornito con una ricca libreria standard che offre strumenti per una vasta gamma di attività, dall'elaborazione di stringhe alla programmazione di rete.
- **Estensibilità:** Python può essere esteso con moduli scritti in altri linguaggi come C o C++, permettendo di ottimizzare le parti critiche per le prestazioni.

Convenzioni

Prima di iniziare a scrivere del codice in Python, è bene apprendere alcune convenzioni che stanno alla base di una scrittura di codice pulito e di facile comprensione:

- **Nomi significativi per variabili e funzioni:** Usare nomi che riflettano lo scopo della variabile o della funzione può rendere il codice più leggibile e manutenibile.
- **PEP 8:** Python ha una guida di stile ufficiale chiamata PEP 8, che fornisce linee guida per la formattazione del codice. Seguire queste linee guida aiuta a mantenere la coerenza tra i progetti Python.
- **Indentazione:** In Python, l'indentazione è semanticamente significativa e determina la struttura del codice. In altre parole, Python usa l'indentazione per definire blocchi di codice. Questo è fondamentale quando si scrivono funzioni, cicli o strutture condizionali. È **fondamentale** mantenere un'indentazione coerente.

- **Commenti e documentazione:** Commentare il codice e mantenere una documentazione aggiornata sono pratiche essenziali. Python supporta le docstring, che sono stringhe di documentazione incorporate nel codice.
- **Principio di Incapsulamento:** Anche se Python non ha modificatori di accesso privati come altri linguaggi, si usa la convenzione di prefissare con un underscore i nomi di attributi e metodi che dovrebbero essere considerati privati.
- **Evitare l'uso eccessivo di globali:** L'utilizzo eccessivo di variabili globali può rendere il codice difficile da debuggare e mantenere. È preferibile limitare il loro uso.
- **Struttura del nome delle costanti:** In Python, le costanti sono tipicamente scritte in lettere maiuscole, con parole separate da underscore, come ad esempio: PI_GRECO.

Seguendo queste convenzioni e sfruttando le potenti caratteristiche di Python, gli sviluppatori possono creare codice efficiente, leggibile e manutenibile.

Guida agli ambienti di sviluppo Python

Un IDE (Integrated Development Environment) è un ambiente di sviluppo che facilita la programmazione. Di seguito vengono presentati i principali ambienti di sviluppo disponibili per Python, con le loro caratteristiche distintive.

Principali IDE per Python

Di seguito vengono elencati alcuni IDE per poter programmare in Python

PyCharm Community Edition

PyCharm Community Edition è un ambiente di sviluppo professionale che offre:

- Completamento automatico del codice intelligente
- Evidenziazione della sintassi
- Debugger integrato
- Interfaccia user-friendly
- Versione gratuita con funzionalità complete

Visual Studio Code

Visual Studio Code con l'estensione Python si caratterizza per:

- Leggerezza e velocità di esecuzione
- Alta personalizzabilità
- Supporto ottimizzato per Python
- Vasta comunità di utenti
- Gratuità e natura open source

Thonny

Thonny è un ambiente particolarmente adatto ai principianti, che presenta:

- Interfaccia semplificata e intuitiva
- Installazione comprensiva di Python
- Debugger visuale con esecuzione passo-passo
- Configurazione ottimizzata per l'apprendimento

IDE Standard

Chiamato ‘IDLE’ (Integrated Development **and Learning** Environment) incluso con Python offre:

- Installazione automatica con Python
- Funzionalità basilari ma complete
- Adeguatezza per progetti semplici
- Assenza di necessità di configurazioni aggiuntive

Guida all'Installazione di Thonny

Per iniziare l'installazione di Thonny è necessario:

- Accedere al sito ufficiale: https://thonny.org
- Selezionare il download appropriato per il sistema operativo in uso

Installazione su Windows

La procedura di installazione su Windows prevede:

- Esecuzione del file .exe scaricato
- Seguire la procedura guidata di installazione
- Confermare i passaggi fino al completamento
- Attendere l'avvio automatico dell'applicazione

Installazione su MacOS

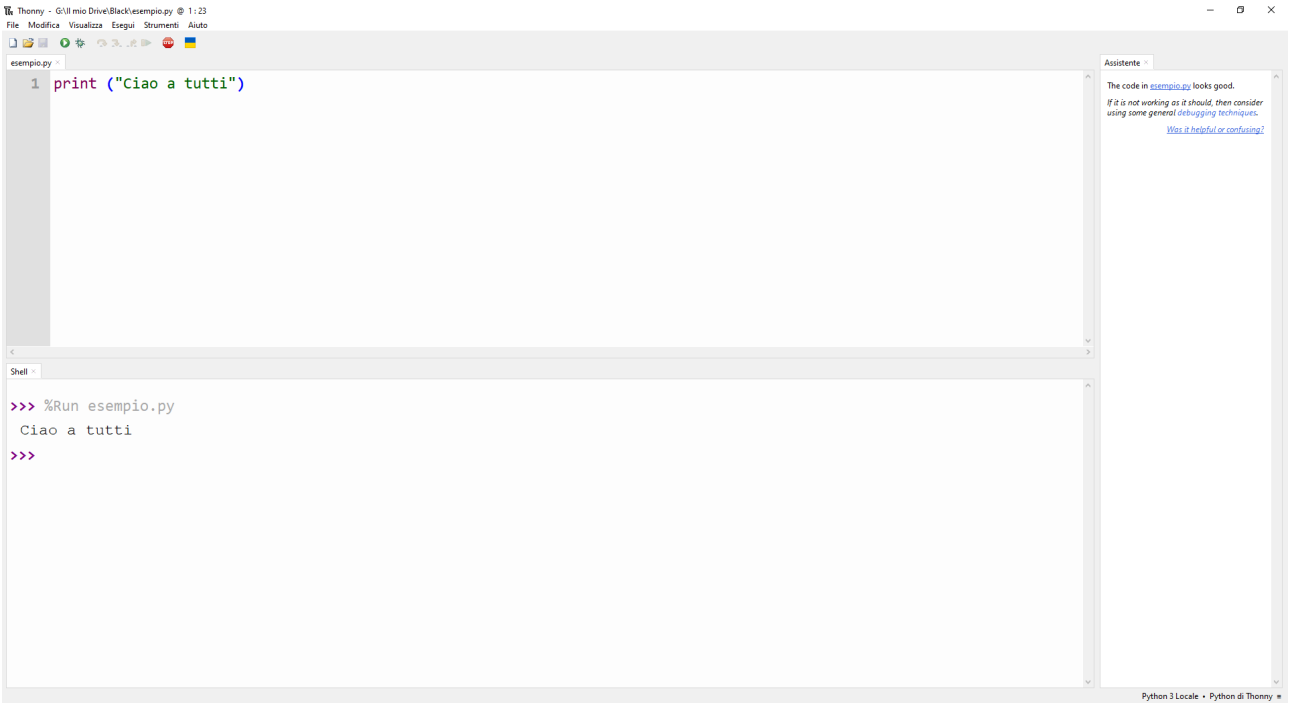
Per MacOS, il processo richiede:

- Apertura del file .pkg scaricato
- Seguire le istruzioni dell'installer
- Localizzare l'applicazione nella cartella Applicazioni
- Configurare eventuali permessi di sicurezza al primo avvio

Ambiente di Lavoro

L'interfaccia di Thonny si compone di:

- Editor di testo nella sezione superiore per la scrittura del codice
- Shell Python nella sezione inferiore per la visualizzazione dei risultati
- Barra degli strumenti con i comandi principali



Sintassi in Python

La sintassi in Python è un insieme di regole che definiscono come scrivere correttamente il codice in modo che possa essere compreso e eseguito dall'interprete Python. Ecco una panoramica delle regole fondamentali di sintassi in Python:

Struttura di Base di un Programma

Un programma Python tipico non richiede una struttura specifica come in C++ o Java. Tuttavia, ecco un esempio di struttura comune:

```
# Importazione di librerie
import math

# Definizione di funzioni
def saluta(nome):
    print(f"Ciao, {nome}!")

# Corpo principale del programma
if __name__ == "__main__":
    saluta("Mondo")
    print("Il valore di pi è circa", math.pi)
```

- **import math** - Questa istruzione importa il modulo math, che fornisce funzioni matematiche.
- **def saluta(nome):** Definisce una funzione chiamata saluta.
- **if __name__ == "__main__":** Questo blocco viene eseguito solo se lo script viene eseguito direttamente (non importato come modulo).

Inoltre, è fondamentale ricordare che a differenza dei linguaggi C, C++ e Java:

- Le stringhe in Python possono essere racchiuse tra virgolette singole o doppie (esempio: 'Ciao' o "Ciao").
- Python usa l'indentazione per definire blocchi di codice, non le parentesi graffe.
- Le istruzioni in Python non richiedono il punto e virgola alla fine.

Editor, interpretazione ed esecuzione di un programma

Si può scrivere il codice Python con un qualsiasi editor di testo semplice. Tuttavia, per una migliore esperienza di programmazione, si consiglia di usare un IDE specifico per Python, come PyCharm, Visual Studio Code con estensioni Python, IDLE (che viene fornito con Python), Spyder, o Jupyter Notebook, che offrono evidenziazione della sintassi, suggerimenti di codice e altre funzionalità utili.

Estensione del File: I file Python vengono salvati con l'estensione *.py.

Esecuzione: Python è un linguaggio interpretato, quindi non c'è una fase di compilazione separata. Per eseguire un programma Python bisogna aprire il terminale o il prompt dei comandi e andare nella directory dove è stato salvato il file (ad esempio prova.py). Quindi, eseguire il comando:

```
python prova.py
```

Questo comando dice all'interprete Python di eseguire lo script prova.py.

Nel caso venga installato un IDE specifico, il programma dopo la prima configurazione iniziale, eseguirà lo script Python tramite apposite funzioni senza passare dal prompt dei comandi.

Python offre anche una modalità interattiva, accessibile digitando semplicemente python nel terminale, che permette di eseguire istruzioni Python una alla volta, utile per test rapidi e apprendimento.

Programma di esempio in Python

```
# Questo è un commento su una singola linea

"""
Questo è un commento
su più righe
"""

print("Ciao a tutti!!!")  # Stampa "Ciao a tutti!!!" sullo schermo
```

Spiegazione del Programma

- **Commenti**: In Python, i commenti su una singola riga iniziano con #. Per commenti su più righe, si possono usare tre virgolette """ all'inizio e alla fine del blocco di commento.
- **print("Ciao a tutti!!!")** - Questa funzione incorporata stampa il messaggio "Ciao a tutti!!!" sullo schermo. Python fornisce funzioni di base come print() senza necessità di importare librerie aggiuntive come avviene in altri linguaggi di programmazione.
- **Struttura del programma:** Python esegue le istruzioni nell'ordine in cui appaiono, dall'alto verso il basso. Non è necessaria una funzione principale esplicita per l'esecuzione del programma come avviene in altri linguaggi di programmazione.
- **Sintassi delle istruzioni:** In Python, le istruzioni non richiedono un carattere speciale di terminazione. Ogni nuova riga viene interpretata come una nuova istruzione.

Variabili

Le variabili in Python sono riferimenti dinamici a oggetti in memoria. L'oggetto è l'effettiva area di memoria che contiene i dati.
In Python, i nomi delle variabili devono seguire queste regole:

- I nomi delle variabili in Python devono essere una sequenza continua di caratteri, senza spazi. (quindi ad esempio “conta pari” deve diventare “conta_pari” oppure “contaPari”);
- Devono iniziare con una lettera (a-z, A-Z);
- Iniziare con underscore per variabili private: _variabile_privata;
- Il resto del nome può contenere lettere, numeri e underscore.
- I nomi sono case-sensitive (variabilePersonale e variabilepersonale sono variabili diverse).
- Non possono essere parole chiave di Python (come if, for, while, ecc.).

Esempi di variabili valide in Python:

```
my_variable = 5
count = 0
_private = "Hello"
camelCase = True
```

Dichiarazione e inizializzazione di variabili

Python è un linguaggio a tipizzazione dinamica, il che significa che non è necessario dichiarare esplicitamente il tipo di una variabile. Il tipo viene inferito automaticamente quando si assegna un valore. Ecco alcuni esempi:

```
x = 5  # x è automaticamente un intero
y = "Hello"  # y è automaticamente una stringa
z = 3.14  # z è automaticamente un float
```

Ambito delle variabili

- Variabili globali: dichiarate fuori da qualsiasi funzione
- Variabili locali: dichiarate all'interno di una funzione
- Uso della keyword global per modificare variabili globali all'interno di funzioni

Esempi:

```
x = 10  # Variabile globale

def funzione():
    global x
    x = 20  # Modifica la variabile globale
    y = 5   # Variabile locale
```

Cancellazione di variabili

Per cancellare una variabile si usa il comando del, come ad esempio:

```
del nome  # Rimuove la variabile 'nome'
```

Tipi di Variabili

Python è un linguaggio a tipizzazione dinamica, ma fortemente tipizzato.

Numeri interi (int)

Precisione arbitraria (non c'è limite alla grandezza):

```
a = 5
b = 123456789012345678901234 5
```

Numeri decimali (float)

Seguono lo standard IEEE 754 a doppia precisione:

```
x = 3.16
y = -0.5
z = 2.5e-4  # Notazione scientifica
```

Numeri complessi

Un numero complesso è un numero della forma ***z = a + bi***, dove ***a*** e ***b*** sono numeri reali e ***i*** è l'unità immaginaria. tale che $i^2 = -1$, dove ***a*** è la parte reale, ***b*** la parte immaginaria e ***i*** è l'unità immaginaria.

```
c = 3 + 4j  # j rappresenta la parte immaginaria
```

Stringhe (str)

Sono immutabili e supportano l'indicizzazione[8] e lo slicing[9]

```
testo = "Informatica"
print(testo[0])      # Output: 'I'
print(testo[1:4])    # Output: 'nfo'
```

Su una stringa è possibile usare alcuni metodi (o funzioni) per semplificare la manipolazione:

```
s = "hello, world"
print(s.capitalize())  # Output: 'Hello, world'
print(s.upper())       # Output: 'HELLO, WORLD'
print(s.find('o'))     # Output: 4
print(s.split(','))    # Output: ['hello', ' world']
```

Variabili Booleane (bool)

Sono variabili che possono assumere solamente 2 valori: True e False (notare la maiuscola).

```
x = True
y = False
```

Operatori semplici

Naturalmente, per poter effettuare delle operazioni tra le variabili si devono utilizzare dei caratteri "speciali" che prendono il nome di operatori.

Gli operatori possono essere suddivisi in varie categorie. Di seguito si elencano gli operatori principali e di più comune utilizzo.

In Python, gli operatori giocano un ruolo cruciale permettendo di eseguire operazioni su variabili e valori. Sono divisi in diverse categorie, ognuna con un proprio scopo. Ecco un riassunto degli operatori semplici nelle categorie specificate: relazionali, aritmetici, logici, e di assegnamento.

[8] Il contenuto di una stringa può essere indicizzato, ossia, il primo carattere assume valore 0, il secondo 1, il terzo 2 e così via.

[9] Lo slicing è un'operazione utilizzata in molti linguaggi di programmazione (come Python) per estrarre una porzione di una sequenza, come una stringa o una lista. Si specificano gli indici di inizio e fine della porzione desiderata, con la sintassi: **sequenza[inizio:fine]** dove l'elemento “inizio” è incluso, mentre quello “fine” è escluso.

Operatori Aritmetici

Gli operatori aritmetici vengono usati per eseguire operazioni matematiche come addizioni, sottrazioni, moltiplicazioni, divisioni, e ottenimento del resto di una divisione (modulo).

```
+          addizione
-          sottrazione
*          moltiplicazione
/          divisione (sempre risultato float)
//         divisione intera (arrotonda verso il basso)
%          modulo (resto della divisione)
**         Esponente (potenza)
```

Operatori Relazionali

Gli operatori relazionali confrontano due valori e restituiscono un valore booleano (True o False).

```
>          maggiore di
>=         maggiore o uguale di
<          minore di
<=         minore o uguale di
==         uguale a
!=         non uguale a (diverso da)
```

Operatori Logici

Gli operatori logici vengono utilizzati per combinare due o più condizioni.

```
and        and logico
or         or logico
not        not logico
```

Operatori di Assegnamento

Gli operatori di assegnamento vengono usati per assegnare un valore a una variabile.
L'operazione di assegnamento consente normalmente di assegnare alla stessa variabile valori diversi durante l'esecuzione del programma.
L’assegnamento di base è eseguito attraverso l’operatore “=” che, in modo molto semplice vuol dire “prendi il valore sul lato destro e copialo nel lato sinistro”.
L'operatore di base è =, ma ci sono anche operatori di assegnamento composti che combinano un'operazione aritmetica con l'assegnamento.

=	**Assegnamento, assegna un determinato valore ad una variabile.** Esempio x = 2 oppure x = y
+=	**Assegnamento con addizione** Esempio x += 2 equivale a x = x + 2
-=	**Assegnamento con sottrazione** Esempio x -= y equivale a x = x - y
***=**	**Assegnamento con moltiplicazione** Esempio x *= 2 equivale a x = x * 2
/=	**Assegnamento con divisione** Esempio x /= 3 equivale a x = x / 3
%=	**Assegnamento con modulo (resto di una divisione)** Esempio x %= 3 equivale a x = x % 3
****=**	**Assegnamento con elevamento a potenza** Esempio x **= 2 equivale a x = x ** 2 ossia x^2
//==	**Assegnamento con divisione intera** Arrotondata sempre per difetto. Esempio x //= 3 equivale a x = x // 3 Se x = 7 x // 2 sarà uguale a 3 (e non a 3.5)

Operazioni matematiche complesse

Per eseguire operazioni matematiche complesse in Python, oltre agli operatori aritmetici di base, spesso si fa affidamento a funzioni matematiche fornite dal modulo math della libreria standard Python. Il modulo math fornisce una vasta gamma di funzioni matematiche per operazioni come radici quadrate, esponenziali, logaritmi, trigonometria, e altro ancora. Qui di seguito sono illustrate alcune delle operazioni matematiche complesse che si possono eseguire utilizzando il modulo math.

Importazione del modulo

Per utilizzare le funzioni matematiche, si deve importare il modulo math all'inizio del programma che si sta scrivendo:

```
import math
```

Esempi di Funzioni Matematiche

Potenza: math.pow(base, esponente) calcola la potenza di un numero elevato a un certo esponente.

```
result = math.pow(2.0, 3.0)  # 2^3 = 8
```

Radice Quadrata: math.sqrt(x) calcola la radice quadrata di x.

```
result = math.sqrt(16.0)  # 4
```

Esponenziale: math.exp(x) calcola il valore di e (base dei logaritmi naturali) elevato alla potenza di x.

```
result = math.exp(1.0)  # e^1 = 2.71828...
```

Logaritmo Naturale: math.log(x) calcola il logaritmo naturale di x.

```
result = math.log(2.71828)  # 1
```

Logaritmo in Base 10: math.log10(x) calcola il logaritmo in base 10 di x.

```
result = math.log10(100.0)  # 2
```

Trigonometria: math.sin(x), math.cos(x), math.tan(x) calcolano rispettivamente il seno, il coseno e la tangente di un angolo x espresso in radianti.

```
result = math.sin(math.pi / 2)  # 1 (o molto vicino a 1)
```

Python offre anche costanti matematiche come math.pi per il valore di π e math.e per il valore di e, che possono essere utilizzate direttamente nelle espressioni matematiche.

Commenti

È importantissimo che il codice sia leggibile anche a distanza di tempo e da parte di persone diverse dall'autore (per essere modificabile).

Per questo è necessario inserire nel codice commenti significativi, che spieghino le funzionalità dei vari metodi, le scelte fatte e quant'altro l'autore ritiene utile per la comprensione del programma.

In Python i commenti possono essere a riga singola o su più righe:

Commenti su singola linea

I commenti a riga singola sono preceduti dal simbolo #, quindi tutti i caratteri successivi a # fino a fine linea sono ignorati in fase di esecuzione.

```
# Questo è un commento su singola linea
x = 5  # Anche questo è un commento su singola linea
```

Commenti su più linee

I commenti su più righe in Python sono in realtà stringhe multilinea non assegnate, racchiuse tra triple virgolette (“””) o tripli apici (‘’’). Questi commenti possono estendersi su più righe, rendendoli utili per commentare blocchi di codice o per inserire note esplicative più lunghe.

```
'''
Questo è un commento
che si estende su
più linee
'''
y = 10
```

Oppure:

```
"""
Questo è un altro modo
di scrivere commenti
su più linee
"""
```

Inserimento dei dati da tastiera

In Python, l'inserimento dei dati da tastiera si effettua principalmente tramite la funzione **input()**. Questa funzione legge una linea di input dall'utente come una stringa.

Ecco come funziona:

- La funzione **input()** è incorporata in Python e non richiede l'importazione di alcuna libreria specifica.
- Si può utilizzare **input()** per leggere vari tipi di dati da tastiera, come numeri interi, a virgola mobile, caratteri e stringhe. Tuttavia, **input()** restituisce sempre una stringa, quindi potrebbe essere necessario convertire l'input in altri tipi di dati se necessario.

Ecco un esempio di come leggere un numero intero e una stringa da tastiera:

```
# Leggere un numero intero
numero = int(input("Inserisci un numero: "))

# Leggere una stringa
parola = input("Inserisci una parola: ")

print(f"Hai inserito il numero {numero} e la parola {parola}.")
```

In questo esempio:

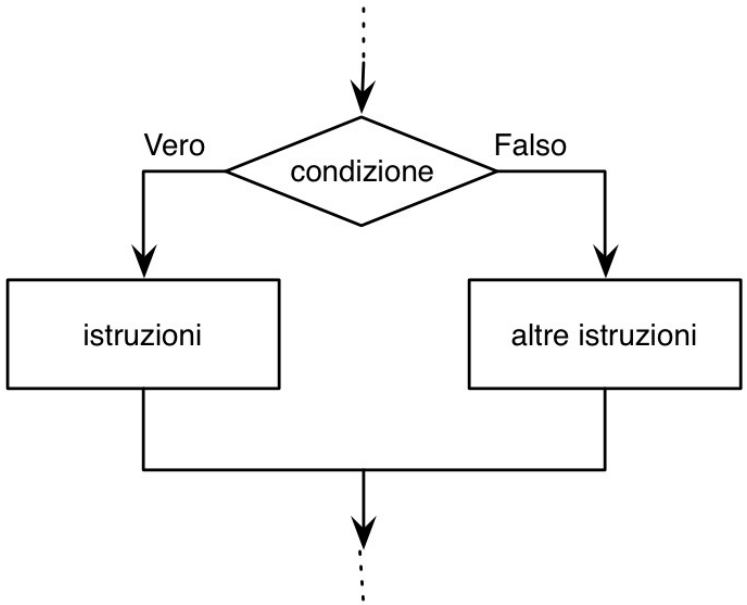
1. **input()** viene utilizzato per richiedere l'input all'utente.
2. Per il numero, si usa **int()** per convertire l'input da stringa a intero.
3. Per la parola, non è necessaria alcuna conversione poiché input() restituisce già una stringa.
4. Si usa una f-string per formattare l'output in modo leggibile.

È importante notare che se l'utente inserisce un valore non valido (ad esempio, una lettera quando si aspetta un numero), Python genererà un errore. In un'applicazione reale, sarebbe opportuno gestire queste situazioni con la gestione delle eccezioni.

Condizionali

I costrutti condizionali (o strutture di controllo) sono delle espressioni che consentono di alterare l’usuale modo di esecuzione di un programma e di eseguire una certa porzione di codice in base ad una “scelta”.

I costrutti condizionali sono quindi semplicemente il modo di tradurre sotto forma di codice algoritmi simili al seguente:

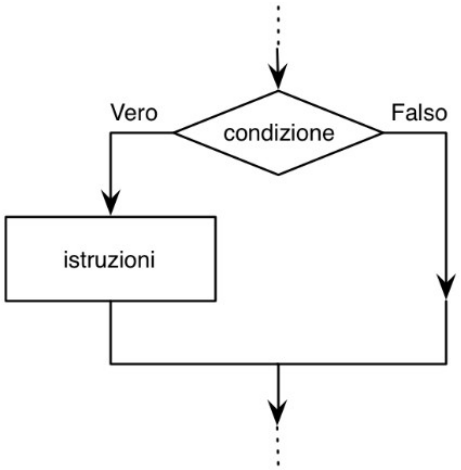


Rappresentazione del diagramma di flusso corrispondente alla condizionale if-else

```
.
.
.
SE condizione1:
    .
    .
    codice da eseguire se condizione1 è soddisfatta
    .
    .
ALTRIMENTI:
    .
    .
    codice da eseguire se la precedente condizione NON è soddisfatta
    .
    .
.
.
.
```

Rappresentazione in pseudo-codice corrispondente alla condizionale if-else

La condizionale “altrimenti” non è obbligatoria, infatti può capitare di eseguire una porzione di codice se si verifica una condizione, nel caso questa condizione non si verifica si prosegue nell’esecuzione del codice successivo.



Rappresentazione del diagramma di flusso corrispondente alla condizionale if

In Python esistono sostanzialmente 2 costrutti condizionali, **if-elif-else** e **match-case** (disponibile da Python 3.10 in poi)

if-elif-else

Tutte le “condizioni” devono essere espressioni di tipo booleane (quindi risultare vere o false).

Tutte le "condizioni" devono essere espressioni che possono essere valutate come booleane (quindi risultare vere o false).
In Python, la struttura base di un costrutto condizionale può includere **if**, **elif** (abbreviazione di "else if"), e **else**.
È importante notare che:

- **if** è sempre richiesto per iniziare un costrutto condizionale.
- **elif** è opzionale, e può essere usato più volte per verificare condizioni aggiuntive.
- **else** è opzionale, e viene usato per catturare tutti i casi che non soddisfano le condizioni precedenti.

La struttura generale può essere:

```
if condizione1:
    # istruzioni se condizione1 è vera
elif condizione2:
    # istruzioni se condizione2 è vera
elif condizione3:
    # istruzioni se condizione3 è vera
else:
    # istruzioni se nessuna delle condizioni precedenti è vera
```

È possibile avere un **if** senza **elif** o **else**:

```
if condizione:
    # istruzioni se la condizione è vera
# il programma continua qui se la condizione è falsa
```

O un if con solo else:

```
if condizione:
    # istruzioni se la condizione è vera
else:
    # istruzioni se la condizione è falsa
```

Esempio che mostra diverse combinazioni:

```
x = 10

if x > 0:
    print("x è positivo")
    if x > 5:
        print("x è maggiore di 5")
    else:
        print("x è positivo ma non maggiore di 5")
elif x < 0:
    print("x è negativo")
else:
    print("x è zero")
```

In questo esempio, si ha un if esterno con un elif e un else, e un altro if-else annidato all'interno del primo blocco if.

match-case

Il costrutto match-case (introdotto in Python 3.10) confronta il valore di una variabile con una serie di pattern possibili (case) ed esegue il blocco di codice corrispondente al "case" che corrisponde al valore della variabile. È spesso usata come alternativa a lunghi blocchi if...elif quando si confrontano molteplici valori di una stessa variabile.

```
match variabile:
    case pattern1:
        # Codice da eseguire se variabile corrisponde a pattern1
    case pattern2:
        # Codice da eseguire se variabile corrisponde a pattern2
    # Si possono aggiungere quanti case si vuole
    case _:
        # Codice da eseguire se nessun case corrisponde
```

NB: In Python, non c'è bisogno di usare "break" alla fine di ogni blocco "case" come in C++ o Java. L'esecuzione non continua automaticamente nei case successivi. Il caso **case _:** è opzionale.

Esempio:

```
x = 3

match x:
    case 1:
        print("Numero 1")
    case 2:
        print("Numero 2")
    case 3:
        print("Numero 3")
    case 4:
        print("Numero 4")
    case _:
        print("Numero diverso da 1 a 4")
```

I costrutti condizionali sono strumenti potenti che consentono di scrivere codice che può prendere decisioni e eseguire percorsi diversi a seconda delle condizioni.

Cicli

I cicli sono strutture di controllo che ripetono un blocco di codice finché una condizione specificata rimane vera. Sono utilizzati per eseguire iterazioni, ovvero per ripetere un'azione per un numero predeterminato di volte o finché una determinata condizione è soddisfatta. Python fornisce diverse strutture di ciclo:

- Ciclo for
- Ciclo while

Ciclo for

In Python, il ciclo for è più versatile rispetto a molti altri linguaggi. Viene utilizzato principalmente per iterare su una sequenza (che può essere una lista, una tupla, un dizionario, un set, o una stringa) o altri oggetti iterabili. La sintassi del ciclo for in Python è la seguente:

```
for elemento in sequenza:
    # istruzioni da eseguire per ogni elemento
```

Esempio:

```
for i in range(5):
    print(i, end=" ")
```

Come risultato verrà stampato a video: ***0 1 2 3 4***
In questo esempio, **range(5)** genera una sequenza di numeri da 0 a 4, e il ciclo for itera su questa sequenza.

Se si vuole un comportamento più simile al ciclo for tradizionale di C++ o Java, si può usare la funzione **range()** con più argomenti:

```
for i in range(0, 10, 2):
    print(i, end=" ")
```

Questo stamperà: ***0 2 4 6 8***
La funzione **range(0, 10, 2)** genera una sequenza che:

- parte da 0;
- arriva fino a 10 (il 10 è escluso);
- si incrementa di 2 ad ogni passo.

Ciclo while

I cicli while in Python funzionano in modo simile a C++ o Java.
Il ciclo viene eseguito fintanto che una condizione viene verificata.

```
while condizione:
    .
    .
    istruzioni da eseguire mentre la condizione è vera
    .
    .
```

Esempio:

```
somma = 0
contatore = 1

while contatore <= 10:
    x = int(input("Inserire un valore per la variabile x: "))
    somma += x
    contatore += 1

print(f"Somma: {somma}")
print(f"Contatore: {contatore}")
```

Come risultato si avrà la somma di 10 numeri inseriti da tastiera, questo perché il ciclo viene eseguito fintanto che la variabile contatore sarà minore o uguale a 10.

Ciclo do-while

Python non ha un equivalente diretto del ciclo do-while come in C++, Java o altri linguaggi di programmazione. Tuttavia, si può simulare un comportamento simile usando un ciclo while con una condizione che viene controllata alla fine del blocco.

Ecco un esempio di come si può implementare un comportamento simile al do-while:

```
while True:
    # istruzioni da eseguire almeno una volta
    if not condizione:
        break
```

Questa struttura garantisce che il blocco di codice venga eseguito almeno una volta, e poi continua l'esecuzione fintanto che la condizione rimane vera.

Esempio:

```
contatore = 0
while True:
    print(f"Il contatore è {contatore}")
    contatore += 1
    if contatore >= 5:
        break
```

Questo codice stamperà i valori del contatore da 0 a 4, eseguendo il blocco almeno una volta anche se la condizione (contatore < 5) fosse falsa all'inizio.

In Python, i cicli for e while offrono grande flessibilità e possono essere utilizzati per gestire tantissime situazioni di iterazione. La scelta tra for e while dipende spesso dalla natura dell'iterazione che si desidera eseguire e dalla chiarezza del codice risultante.

Interruzione e continuazione dei Cicli

I cicli sono uno strumento fondamentale in Python per implementare algoritmi che richiedono ripetizioni di operazioni, permettendo di gestire facilmente compiti che altrimenti richiederebbero molte linee di codice ripetitivo. È utile conoscere due istruzioni:

- **break:** Termina il ciclo e passa il controllo all'istruzione subito dopo il corpo del ciclo.
- **continue:** Salta l'iterazione corrente del ciclo e passa direttamente alla valutazione della condizione per la prossima iterazione.

Queste istruzioni funzionano allo stesso modo sia nei cicli **for** che nei cicli **while**.

Esempio con break e continue:

```
for i in range(10):
    if i == 3:
        continue  # Salta il valore 3
    if i == 8:
        break  # Esce dal ciclo quando i è 8
    print(i, end=" ")
```

Come risultato verrà stampato a video: ***0 1 2 4 5 6 7***.

Poiché:

- Quando **i** è uguale a **3**, l'istruzione **continue** fa saltare al ciclo l'iterazione corrente, passando direttamente alla successiva.
- Quando **i** raggiunge **8**, l'istruzione **break** termina immediatamente il ciclo, senza stampare **8** o **9**.

Esempio con break e continue con registrazione Audio 16 bit:
Supponiamo che i segnali oltre il valore 25000, in valore assoluto, siano distorti, per cui vogliamo scartarli.
Supponiamo inoltre che la funzione segnale_audio_in_ingresso, ritorni valori da -32000 a +32000, e che in assenza di segnale il valore ritornato sia -32768

```
i = 0 # Indice iniziale per la registrazione
while (True):
    s = segnale_audio_in_ingresso()

    if abs(s) >= 25000: # Il valore è distorto
        continue  # Salta all iterazione successiva
    if s == -32768: # Assenza di segnale
        break  # Esci dal ciclo

    registrazione[i] = s

    i = i + 1

print("Registrazione terminata")
```

Funzioni

Una funzione in Python è un blocco di codice riutilizzabile che esegue una specifica operazione.

Un principio fondamentale della programmazione è il concetto di riutilizzo del codice. Generalmente, più un programma diventa complesso e strutturato, più saranno le righe di codice che lo compongono. Per garantire che il codice rimanga snello e più leggero possibile, è essenziale evitare ripetizioni superflue richiamando blocchi di codice attraverso le cosiddette funzioni.

In questo modo si può riutilizzare un blocco di codice in molte parti del programma, semplicemente richiamando il nome con cui è stato definito, senza dover riscrivere tutto il blocco ogni volta.

Le funzioni permettono di strutturare i programmi in segmenti di codice riutilizzabili e gestibili, facilitando la lettura, la scrittura e la manutenzione del codice.

Struttura di una funzione

La definizione di una funzione in Python inizia con la parola chiave **def**, seguita dal nome della funzione, una lista opzionale di parametri tra parentesi, e i due punti. Il corpo della funzione deve essere indentato. Ecco la sintassi generale:

```
def nome_della_funzione (parametro/i in ingresso):
      .
      .
      istruzioni
      .
      .
      return …
```

Ogni funzione può:

- ricevere in ingresso nessuno, uno o più parametri
- restituire uno o più valori (usando il **return**). Python permette di restituire anche tuple e/o liste, consentendo quindi il ritorno di più valori contemporaneamente
- non restituire nulla (in questo caso il **return** è opzionale, quindi se omesso, la funzione restituirà implicitamente **None**)

Esempio di funzione con 2 parametri in ingresso che restituisce la somma di un numero intero:

```
def somma_numeri(numero1, numero2):
    risultato = numero1 + numero2
    return risultato  # elemento da restituire
```

Esempio di utilizzo (chiamata) di una funzione:

```
totale = somma_numeri(5, 3)      # totale = 8
```

Esempi di funzioni

Di seguito vengono elencati alcuni esempi, di come può essere strutturata una funzione, sia nella costruzione, sia nella chiamata della stessa con il relativo output

Funzioni senza parametri in ingresso

Questa è la struttura più semplice: la funzione si chiama **nome_funzione**, non prende parametri (parentesi vuote) e quando viene chiamata stampa solo "Ciao!".

```
def nome_funzione():
    # Blocco di codice
    print("Ciao!")
```

Richiamando questa funzione:

```
print(nome_funzione())
```

L’output a video sarà:

```
Ciao!
```

Funzioni con parametri in ingresso

Questa funzione prende un parametro **nome** e lo usa nel messaggio. La **f** prima delle virgolette permette di inserire variabili nel testo usando **{}**.

```
def saluta(nome):
    print(f"Ciao {nome}!")
```

Richiamando questa funzione:

```
saluta("Pietro")
```

L’output a video sarà:

```
Ciao Pietro!
```

Da notare che se si omette la lettera **f**, non è possibile inserire il parametro durante la chiamata della funzione.

```
def saluta(nome):
    print("Ciao {nome}!")
```

Richiamando questa funzione:

```
saluta("Pietro")
```

L’output a video sarà:

```
Ciao {nome}!
```

Funzioni con valore di ritorno

Questa funzione prende due parametri, li somma e restituisce il risultato con **return**. Il valore può essere salvato in una variabile.

```
def somma(a, b):
    return a + b
```

Richiamando questa funzione:

```
risultato = somma (3, 4)
print("Il risultato della somma è",risultato)
```

L’output a video sarà:

```
Il risultato della somma è 7
```

Funzioni con parametri predefiniti

Questa funzione, se non si specifica il parametro **nome**, usa "Amico" come valore predefinito.

```
def saluta_persona(nome="Amico"):
    print(f"Buongiorno {nome}")
```

Richiamando questa funzione:

```
saluta_persona()
saluta_persona("Pietro")
```

L’output a video sarà:

```
Buongiorno Amico
Buongiorno Pietro
```

Da notare che se non si specifica il parametro di default stamperà “Amico”

Funzioni con numero variabile di argomenti

L'asterisco ***** permette di passare qualsiasi numero di parametri. La funzione **sum** è una funzione incorporata di Python che permette di fare la somma tra più numeri.

```
def somma_numeri(*numeri):
    totale = 0
    for num in numeri:
        totale += num
    return total
```

Richiamando questa funzione:

```
print(somma_numeri(2, 3))
print(somma_numeri(2, 3, 5))
print(somma_numeri(2, 3, 5, 7))
print(somma_numeri(2, 3, 5, 7, 9))
```

L’output a video sarà:

```
5
10
17
26
```

Funzioni incorporate (built-in)

Le funzioni incorporate sono funzioni "pre-costruite" che Python include automaticamente che si possono usare subito senza dover scrivere il codice o importare librerie esterne.

Di seguito vengono elencate le principali funzioni incorporate all’interno di Python:

```
# Conversioni di tipo di variabili
int()      # converte in intero: int("123") → 123
float()    # converte in decimale: float("12.3") → 12.3
str()      # converte in stringa: str(123) → "123"
bool()     # converte in booleano: bool(1) → True
list()     # converte in lista: list("abc") → ['a','b','c']

# Funzioni matematiche
abs()      # valore assoluto: abs(-5) → 5
max()      # massimo: max([1,2,3]) → 3
min()      # minimo: min([1,2,3]) → 1
sum()      # somma: sum([1,2,3]) → 6
round()    # arrotonda: round(3.7) → 4

# Funzioni di input/output
print()    # stampa a schermo
input()    # legge input da tastiera
map()      # applica una funziona agli elementi di una lista
```

Esempio funzione map()

```
a = [1, 2, 3, 4]


# Funzione applicata
def raddoppia(val):
  return val*2

res = list(map(raddoppia, a))
print(res)
```

Il risulato sarà:

[2, 4, 6, 8]

Liste

Una lista in Python è una struttura dati che può contenere una sequenza ordinata di elementi. A differenza degli array tradizionali presenti in altri linguaggi di programmazione, le liste Python sono più flessibili perché:

- Possono contenere elementi di tipi diversi (numeri interi, con la virgola, stringhe, ecc..)
- Sono dinamiche (non sono di una grandezza predefinite, di conseguenza la loro dimensione può crescere o diminuire)
- Offrono molte funzioni incorporate per la manipolazione (aggiunta, eliminazione o modifica dei suoi elementi)

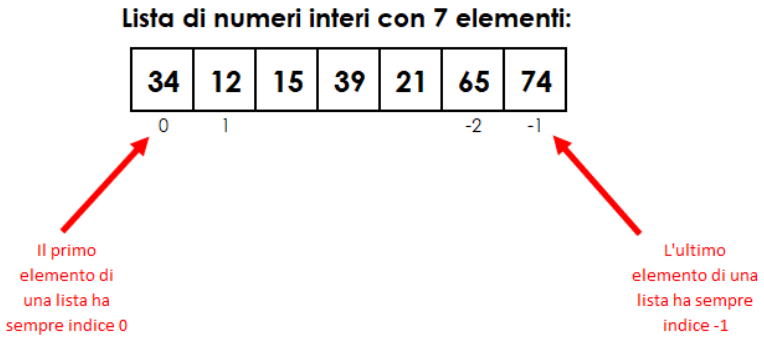
Creazione di una lista

Gli elementi di una lista sono sempre racchiusi tra parentesi quadre [ ]. Le liste sono indicizzate, con il primo elemento all'indice 0. Per creare una lista basta dargli un nome e inserire tra parentesi quadre i suoi elementi, come nell’esempio sottostante:

```
# Liste possono essere create in diversi modi:
vuota = []
numeri = [1, 2, 3, 4, 5]
stringhe = ["mela", "pera", "banana"]
mista = [1, "ciao", 3.14, True]
matriosca = [1, [2, 3], [7, 5, 9]]  # liste con all’interno altre liste
```

Accesso agli elementi di una lista

Una lista, indipendentemente dal tipo di dati di cui è composta, può essere manipolata considerando che il **primo elemento** di una lista ha **indice 0**, il secondo 1 e così via, mentre invece l’**ultimo elemento** ha **indice -1**, il penultimo -2 e così via.



Esempio:

```
numeri = [10, 20, 30, 40, 50]

# Indici positivi (da sinistra, partendo da 0)
primo = numeri[0]    # 10
secondo = numeri[1]  # 20

# Indici negativi (da destra, partendo da -1)
ultimo = numeri[-1]  # 50
penultimo = numeri[-2]  # 40

# Slicing (estrarre porzioni)
primi_tre = numeri[0:3]    # [10, 20, 30]
```

```
da_secondo = numeri[1:]     # [20, 30, 40, 50]
fino_terzo = numeri[:3]     # [10, 20, 30]
pari = numeri[::2]          # [10, 30, 50]
```

Accesso agli elementi di una lista all’interno di un’altra lista

Prendendo in considerazione il seguente esempio:

```
lista = [[1, [23, 37], [74, 53, 96]]]

# Accesso al numero 1
print(lista[0][0])     # Output: 1

# Accesso ai numeri [23, 37]
print(lista[0][1][0]) # Output: 23
print(lista[0][1][1]) # Output: 37

# Accesso ai numeri [74, 53, 96]
print(lista[0][2][0]) # Output: 74
print(lista[0][2][1]) # Output: 53
print(lista[0][2][2]) # Output: 96
```

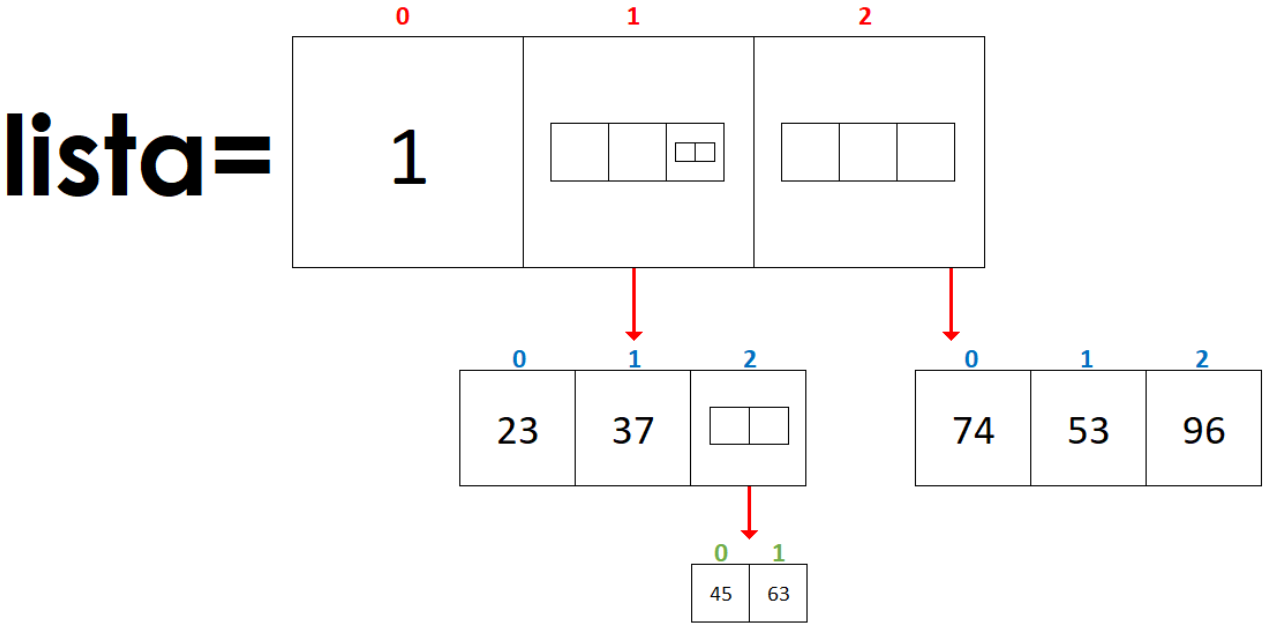
Spiegazione degli indici:

- lista[0] accede alla prima lista esterna
- Il secondo indice ([0], [1] o [2]) seleziona quale elemento della sottolista
- Il terzo indice, quando presente, accede ai numeri dentro le liste annidate

Prendendo in considerazione quest’altro esempio:

```
lista = [1, [23, 37, [45, 63]], [74, 53, 96]]
```

La struttura è così formata:



Di conseguenza, per accedere agli elementi della lista e delle sotto-liste contenute, si accede nel modo seguente:

```
# Accesso al primo elemento
print(lista[0])        # Output: 1

# Accesso al secondo gruppo [23, 37, [45, 63]]
print(lista[1][0])     # Output: 23
print(lista[1][1])     # Output: 37
print(lista[1][2][0]) # Output: 45
print(lista[1][2][1]) # Output: 63

# Accesso al terzo gruppo [74, 53, 96]
print(lista[2][0])     # Output: 74
print(lista[2][1])     # Output: 53
print(lista[2][2])     # Output: 96
```

Modifica delle liste

Le seguenti funzioni permettono di aggiungere o rimuovere elementi all’interno di una lista

Esempio:

```
numeri = [1, 2, 3, 4, 5]

# Modificare elementi esistenti
numeri[0] = 10

# Aggiungere elementi
numeri.append(6)         # aggiunge alla fine
numeri.insert(1, 20)     # inserisce in posizione specifica
numeri.extend([7, 8, 9])  # aggiunge più elementi

# Rimuovere elementi
numeri.remove(3)         # rimuove il primo 3 trovato
del numeri[0]           # rimuove per indice
ultimo = numeri.pop()    # rimuove e restituisce l'ultimo
numeri.clear()          # svuota la lista
```

Operazioni comuni

Le seguenti funzioni permettono di ordinare la lista o di trovare particolari valori

Esempio:

```
numeri = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]

# Ordinamento
numeri.sort()            # ordina la lista originale
numeri_ordinati = sorted(numeri)  # crea nuova lista ordinata
numeri.reverse()         # inverte l'ordine

# Ricerca
conta = numeri.count(1)     # conta occorrenze di 1
indice = numeri.index(4)    # trova posizione di 4

# Altre operazioni
lunghezza = len(numeri)       # numero di elementi
lista1 = [None]*len(numeri)  # crea una lista con la dimensione uguale
ad un'altra lista
lista2 = [None]               # crea una lista vuota
```

Algoritmi di ricerca

In Python, gli algoritmi di ricerca vengono utilizzati per trovare elementi specifici all'interno di una struttura dati, come una lista o un array. Due algoritmi di ricerca comuni sono la ricerca lineare e la ricerca binaria.

1. **Ricerca Lineare** (Sequential Search):
La ricerca lineare esamina ogni elemento della lista uno per uno, fino a quando non trova l'elemento cercato o finché non ha esaminato tutti gli elementi. È semplice da implementare, ma inefficiente su liste lunghe, in quanto ha una complessità temporale di **O(n)**.

Esempio di implementazione in Python:

```
def ricerca_lineare(lista, elemento_da_cercare):
    for indice, elemento in enumerate(lista):
        if elemento == elemento_da_cercare:
            return indice  # Restituisce l'indice dell'elemento trovato
    return -1  # Restituisce -1 se l'elemento non è trovato
```

Ricerca Binaria:
La ricerca binaria è più efficiente della ricerca lineare, ma funziona solo su liste ordinate.
Divide la lista in due metà (metodo **divide et impera**) e verifica quale metà contiene l'elemento cercato, ripetendo questo processo fino a quando l'elemento non è trovato o la lista non è vuota. Ha una complessità temporale di $\mathbf{O(\log_2 n)}$.

Esempio di implementazione in Python:

```
def ricerca_binaria(lista_ordinata, elemento_da_cercare):
    basso = 0
    alto = len(lista_ordinata) - 1
    while basso <= alto:
        mezzo = (basso + alto) // 2  # Calcola l'indice del mezzo
        if lista_ordinata[mezzo] == elemento_da_cercare:
            return mezzo  # Restituisce l'indice dell'elemento trovato
        elif lista_ordinata[mezzo] < elemento_da_cercare:
            basso = mezzo + 1  # La metà superiore contiene l'elemento
        else:
            alto = mezzo - 1  # La metà inferiore contiene l'elemento
    return -1  # Restituisce -1 se l'elemento non è trovato
```

Svolgimento ricerca binaria:

```
Numero da cercare: 15

3  4 15 22 26 40 45 50 57 60   VETTORE V di N posizioni, N = 10
0  1  2  3  4  5  6  7  8  9
```

PASSO 1:

```
0           4              9
V[4] < 15 ?  -> 26 < 15 ? NO
Nuovo limite alto: 4-1 -> 3
```

PASSO 2:

```
0  1     3
V[1] < 15?  -> 4 < 15 ? SI
Nuovo limite basso: 1+1 -> 2
```

PASSO 3:

```
      2  3
15 == 15? SI
```

PASSO 4:

Indice trovato: 2

Efficienza algoritmica

Efficienza algoritmica si riferisce alla capacità di un algoritmo di utilizzare le risorse computazionali in modo ottimale per risolvere un problema. In pratica, un algoritmo efficiente è quello che riesce a completare il compito richiesto nel minor tempo possibile, con la minore quantità di memoria utilizzata.

L'efficienza algoritmica è un concetto fondamentale in informatica e in particolare nella progettazione di algoritmi. Un algoritmo efficiente è caratterizzato da due aspetti principali:

1. **1. Complessità temporale:**
   Misura il tempo necessario per eseguire l'algoritmo in relazione alle dimensioni dell'input. Un algoritmo con complessità temporale inferiore (ad esempio, polinomiale) è considerato più efficiente di uno con complessità esponenziale.

- **2. Complessità spaziale:**

Misura la quantità di memoria che l'algoritmo utilizza durante la sua esecuzione. Un algoritmo che utilizza meno memoria è considerato più efficiente.

Importanza dell'efficienza algoritmica:

- **Riduzione dei tempi di esecuzione:**
  Un algoritmo efficiente permette di risolvere problemi più rapidamente, migliorando la performance complessiva di un sistema.

- **Riduzione del consumo di risorse:**

Utilizzando meno risorse, un algoritmo efficiente contribuisce a ridurre il consumo energetico e a migliorare la scalabilità del sistema.

- **Ottimizzazione dei processi:**

L'efficienza algoritmica permette di progettare sistemi più efficienti e di ottimizzare processi che richiedono un'elevata quantità di calcoli.

Esempi:

Ricerca lineare: complessità temporale **O(n) (lineare)**
Ricerca binaria: complessità temporale **O($\log_2$ n) (logaritmica)**.
In questo caso, la ricerca binaria è più efficiente, soprattutto per elenchi molto grandi.

Si usa la lettera **O**, che sta per ‘**ordine di grandezza**’. In **inglese** è chiamata ‘**Big O Notation**’.

Spiegazione efficienza ricerca lineare

Il ciclo itera nel caso peggiore **N volte**, dove N é la lunghezza dell’array, per cui la complessita é **O(N)**

Spiegazione efficienza ricerca binaria

Vettore 100 posizioni.

100→ 50→ 25→ 13→ 7→ 4→ 2→ 1

A ogni passo si esclude meta delle posizioni, per cui dividendo per due a ogni passo stiamo faciendo il contrario di una crescita esponenziale, ovvero una decrescita logaritmica, in base 2.

$\log_2(100) = \ln(100)/\ln(2) = 4.605 / 0.693 = 6.645 \sim 7$ passi

Se N=100 e il numeri di passi (sequenza di operazioni) e’ $\log_2(N)$, si dice che **l’efficienza** dell’algoritmo é:

$O(\log_2(N))$

Nel calcolo della complessita si calcola solo il caso peggiore, ovviamenti ci sono molti casi fortunati in cui il numero di passi effettivamente eseguiti sono inferiori.

Esempio:

Ricerca lineare: sequenza : 3 5 12 20 40 , se sto cercando il numero 3, lo trovero al primo passo, cio non toglie che il caso peggiore e’ quando cerco 40, per il quale mi servono 5 passi, che coincidono con la complessita: O(N) , con N=5 lunghezza del vettore.

Ricerca binaria: sequenza 3 5 12 20 40, se sto cercando 12, al primo passo la ricerca binaria testa il valore mediano, 12, che gia coincide con quello cercato, per cui lo trovo con un solo passo. Ma è un caso fortunato, e la complessita deve esprimere il caso peggiore. Il caso peggiore è per esempio (ci sono 2 casi peggiori qui) quando cerco 1:

passo 1: testo il mediano 12, restringo il range a (3, 12)

passo 2: testo il mediano 5, restringo il range a (3, 5)

passo 3: testo il mediano 3: trovato 1

passi usati 3, $O(\log_2 N)$ con N=5 , $\log_2(5) = 2.32 \sim 3$

Ricorsione

Codice Python

Fattoriale: N! = 1 * 2 * … * N = N * … * 2 * 1

Esempi:

- 2! = 1 * 2 = 2
- 3! = 1* 2 * 3 = 6
- 4! = 1 * 2 * 3 * 4 = 24

```
def fattoriale(n):
  if (n<2):
    return 1
  return n*fattoriale(n-1)


y = fattoriale(5)
print(y)
```

Esecuzione di esempio per n = 3

```
fattoriale(3)
→
       return 3*fattoriale(3-1)
       →
              fattoriale(2)
              →
                     return 2*fattoriale(2-1)
                     →
                            fattoriale(1)
                            →
                                   return 1
                            ←
                     return 2 * 1
              ←

       return 3 * 2
←

Valore ritornato dalla funzione: 6
```

Tuple

In Python, oltre alle liste, ci sono le tuple.

Sono sequenze di valori ordinate, nel senso che ogni elemento di una tupla ha un indice, che va da 0 fino alla dimensione della tupla – 1.

A differenza delle liste, le tuple una volta create non possono essere modificate, usano quindi meno risorse e sono piú efficienti.

Per creare una tupla, la sintassi é la stessa delle liste, ma usa parentesi tonde invece che quadre.

Esempio:

```
nipoti_di_paperino = (“Qui”, “Quo”, “Qua”)

print(paperino[1])
```

Output:

```
Quo
```

Operazioni con tuple

Concatenazione:

```
a = (1, 2, 3)

b = (4, 5, 6)

c = a + b

print(c)
```

Output:

```
(1, 2, 3, 4, 5, 6)
```

Conteggio numerosita di un elemento contenuto:

```
a = (1, 2, 2, 3, 2)

print(a.count(2))
```

Output:

```
3
```

Conversione di una tupla in una lista:

```
a = (1,2,3)

lista_a = list(a)
```

Conversione di una lista in una tupla:

```
a = [1,2,3]

tupla_a = tuple(a)
```

Insiemi

Un insieme non ha valori duplicati, a differenza di una lista o una tupla, che può averli.

Gli elementi di un insieme

- **non sono ordinati,** ovvero ogni volta che che un insieme viene usato potrebbe mostrare i valori in un ordine ogni volta diverso
- non **sono modificabili,** ma possono essere rimossi, e altri ne possono essere aggiunti.

Operazioni con insiemi:

Unione

```
a = {3, 5, 8}

b = {8, 2}

c = a | b

print(c)
```

Output:

```
{2, 3, 5, 8}
```

Intersezione

```
a = {3, 5, 8}

b = {8, 2}

c = a & b
```

```
print(c)
```

Output:

```
{8}
```

Rimozione elementi

```
a = {3, 5, 8}

a.remove(5)
```

Conversioni in lista o tupla

```
a = {1,2,3}

lista_a = list(a)

tupla_a = tuple(a)
```

Conversioni di liste o tuple in insieme

Convertire una lista o di una tupla in un insieme, causa la perdita dei valori ripetuti.

Esempio:

```
lista = [ 1, 2, 3, 2, 4, 3, 5, 5, 5]

tupla = (20, 30, 30, 45, 20)

insieme_da_lista = set (lista)

insieme_da_tupla = set (tupla)

print(lista)

print(tupla)

print(insieme_da_lista)

print(insieme_da_tupla)
```

Output:

```
[1, 2, 3, 2, 4, 3, 5, 5, 5]

(20, 30, 30, 45, 20)

{1, 2, 3, 4, 5}

{20, 45, 30}
```

Dizionari

Gli elementi di un dizionario sono **ordinati**, **modificabili** e **non consentono duplicati**.

Gli elementi di un dizionario sono presentati in coppie chiave: valore e possono essere consultati utilizzando il nome della **chiave**.

```
dizionario_persona = {
    "Nome": "Antonio",
    "Cognome": "Rossi",
    "Anno": 1970
}
dizionario_termini_icd = {
    "polmonite batterica": "J15.9",
    "diabete mellito tipo 2": "E11.9",
    "scompenso cardiaco": "I50.9"
}
print(dizionario_persona)
print(dizionario_termini_icd)
```

Accesso ai valori tramite parola chiave

```
print(dizionario_persona["Cognome"])
print(dizionario_termini_icd["scompenso cardiaco"])
```

Modo alternativo di creare n dizionario

```
dizionario_persona = dict(Nome = "Antonio", Cognome = "Rossi", Anno = 1970)
print(dizionario_persona)
```

Listare le chiavi

```
x = dizionario_persona.keys()
print(x)
```

Listare i valori

```
x = dizionario_persona.values()
print(x)
```

Listare le coppie chiave:valore

```
x = dizionario_persona.items()
print("items:", x)
```

Cambiare un valore

```
dizionario_persona["Cognome"] = "Bianchi"
print(dizionario_persona)
```

Controllare esistenza di una chiave

```
if "Anno" in dizionario_persona:
  print("'Anno' e' ua chiave del dizionario_persona")
```

Aggiungere valori

```
dizionario_persona.update({"Citta": "Milano"})
print(dizionario_persona)
```

Rimozione valori

```
dizionario_persona.pop("Citta")
print(dizionario_persona)
```

Loop dizionario

```
for x in dizionario_persona:
    print(x) # Mostra la chiave
    print(dizionario_persona[x])  # Mostra il valore


for x,y in dizionario_persona.items():
    print(x, y)
```

Copia

```
copia = dizionario_persona.copy()
riferimento = dizionario_persona


dizionario_persona.pop("Anno")
print("copia:", copia)
print("riferimento:", riferimento)
```

Fondamenti sulle matrici

In matematica, una **matrice** è una griglia rettangolare di numeri.

Ogni elemento è identificato con due indici, tipicamente i, j.

Matrice A =

1	3	9	2
4	7	3	8
8	6	7	5

Se la matrice si chiama A, l'elemento posizionato nella riga i e nella colonna j può essere indicato in vari modi: ad esempio come Ai,j, o tramite parentesi quadre A[i,j].
Vettori colonna e vettori riga sono tipi particolari di matrici.

Somma
Due matrici A e B possono essere sommate se hanno le stesse dimensioni.
La loro somma A + B è definita come la matrice i cui elementi sono ottenuti sommando i corrispettivi elementi di A e B.

Formalmente: $(A + B)_{i,j} := A_{i,j} + B_{i,j}$

L'elemento di posizione (i,j) in C è dato dalla somma di $A_{i,j}$ e $B_{i,j}$

$$\underset{A}{\begin{bmatrix} 1 & 4 \\ 3 & 1 \\ 2 & 5 \end{bmatrix}} + \underset{B}{\begin{bmatrix} 6 & 0 \\ 2 & 8 \\ 7 & 3 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 7 & 4 \\ 5 & 9 \\ 9 & 8 \end{bmatrix}}$$

La moltiplicazione per uno scalare è un'operazione che, data una matrice A ed un numero c (detto scalare), costruisce una nuova matrice cA, il cui elemento è ottenuto moltiplicando l'elemento corrispondente di A per c.
Formalmente: $(cA)_{ij} := cA_{i,j}$

TODO aggiungere immagine

Prodotto tra matrici A×B

È definito soltanto se il numero di righe di B coincide con il numero n di colonne di A. Il risultato è una matrice con lo stesso numero di righe di A e lo stesso numero di colonne di B.

Se A è una matrice m×n e B una matrice n×p, C = A×B sarà una matrice m×p.

Ogni elemento C_{ij} della matrice finale, si calcola come somma dei prodotti tra la riga **i** di a e la colonna j di **B**,

$C_{ij} = A_{i0} * B_{0j} + A_{i1} * B_{1j} + A_{i2} * B_{2j} + A_{i3} * B_{3j}$

Esempio:

Matrice A (3×2):

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Matrice B (2×4):

$$B = \begin{bmatrix} 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 \end{bmatrix}$$

$$C = A \times B = \begin{bmatrix} (1 \times 7 + 2 \times 11) & (1 \times 8 + 2 \times 12) & (1 \times 9 + 2 \times 13) & (1 \times 10 + 2 \times 14) \\ (3 \times 7 + 4 \times 11) & (3 \times 8 + 4 \times 12) & (3 \times 9 + 4 \times 13) & (3 \times 10 + 4 \times 14) \\ (5 \times 7 + 6 \times 11) & (5 \times 8 + 6 \times 12) & (5 \times 9 + 6 \times 13) & (5 \times 10 + 6 \times 14) \end{bmatrix}$$

Come vedete, la matrice risultante C ha dimensioni 3x4, ovvero il num di righe di A x il num di colonne di B.

$$C = \begin{bmatrix} (7 + 22) & (8 + 24) & (9 + 26) & (10 + 28) \\ (21 + 44) & (24 + 48) & (27 + 52) & (30 + 56) \\ (35 + 66) & (40 + 72) & (45 + 78) & (50 + 84) \end{bmatrix} = \begin{bmatrix} 29 & 32 & 35 & 38 \\ 65 & 72 & 79 & 86 \\ 101 & 112 & 123 & 134 \end{bmatrix}$$

Matrici quadrate

Una matrice si dice **quadrata** se ha lo stesso numero di righe e colonne.

Una matrice quadrata ha una **diagonale principale**, quella formata da tutti gli elementi $A_{i,j}$ con indici uguali i=j. La somma di questi elementi è chiamata **traccia**.

L'operazione di **trasposizione** trasforma una matrice quadrata A nella matrice A^t ottenuta scambiando ogni $A_{i,j}$ con $A_{j,i}$, in altre parole ribaltando la matrice intorno alla sua diagonale principale.

Una matrice tale che $A_{i,j} = A_{j,i}$ è una matrice **simmetrica**. In altre parole, A è simmetrica se $A = A^t$.

Se tutti gli elementi che non stanno nella diagonale principale sono nulli, la matrice è detta **diagonale**.

Python

Consideriamo la seguente matrice

1	3	9	2
4	7	3	8
8	6	7	5

Possiamo rappresentarla come lista di liste

```
M = [ [1,3,9,2],
      [4,7,3,8],
      [8,6,7,5] ]
```

Generazione matrice con zeri

Metodo senza inizializzazione matrice, appendendo liste 'colonna' contenenti 0 a una lista vuota di righe

```
numerorighe = 3
numerocolonne = 4
M = [] # lista righe vuota
for i in range(numerorighe):
    N = [] # lista valori riga
    for j in range(numerocolonne):
        N.append(0)
    M.append(n)
```

In pratica ad ogni iterazione di for j, viene creata una lsita N con i valori di una riga (tutti zeri in questo esempio)

```
[0,0,0]
```

e viene appesa alla lista M, inizialmente vuota:

```
Passo 1: [[0, 0, 0, 0]]
Passo 2: [[0, 0, 0, 0], [0, 0, 0, 0]]
Passo 3: [[0, 0, 0, 0], [0, 0, 0, 0],  [0, 0, 0, 0]]
```

Che mostrata in verticale é chiaramente una matrice rettangolare 3 x 4:

```
 [[0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 0, 0]]
```

Metodo diretto definendo una matrice iniziale piena di zeri.

```
numerorighe = 3
numerocolonne = 4
M = [ [0,0,0,0],
       [0,0,0,0],
       [0,0,0,0]]
```

Inserimento numeri da terminale

Metodo con append

```
numerorighe=3
numerocolonne=4
M=[]
for i in range(numerorighe):
    N=[]
    for j in range(numerocolonne):
        x = int(input("Inserire-elemento i:" + str(i) +  str(" j:") + str(j)))
        N.append(x)
    M.append(N)


print(M)
```

Metodo con indici i,j:

```
numerorighe=3
numerocolonne=4
M=[[0,0,0,0],
   [0,0,0,0],
   [0,0,0,0]]
for i in range (numerorighe):
    for j in range(numerocolonne):
        M[i][j] = int(input("Inserire-elemento i:" + str(i) +  str(" j:") + str(j)))


print(M)
```

Controllo Simmetria

```
dimensione = 4
m = [[1, 2, 3, 4],
     [2, 3, 5, 0],
     [3, 5, 6, 8],
     [4, 0, 8, 0]]


simmetrica = True
```

```
for i in range(dimensione):
    for j in range(dimensione):
        if m[i][j] != m[j][i]:
            simmetrica = False


if simmetrica:
    print("La matrice è simmetrica")
else:
    print("La matrice non è simmetrica")
```

Somma di matrici

```
M1 = [ [2,5,0],
       [0,1,8],
       [3,5,7]]
M2 = [ [10,10,10],
       [20,20,20],
       [30,30,30]]

M3 = [ [0,0,0],
       [0,0,0],
       [0,0,0]]

dim = 3 # Matrici quadrate 3 x 3 ovvero 3 righe x 3 colonne

for i in range(0,dim):
    for j in range(0,dim):
        M3[i][j] = M1[i][j] + M2[i][j]
```

Prodotto vettoriale tra matrici

```
M1 = [[1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]]


M2 = [[5, 3, 9],
      [7, 4, 2],
      [6, 1, 4]]


C =[[0,0,0],
    [0,0,0],
    [0,0,0]]


SIZE = 3


for i in range(SIZE):
    print (M1[i])


print()


for i in range(SIZE):
    print (M2[i])


def product(A,B,C):
    for i in range (SIZE):
        for j in range(SIZE):
            for k in range (SIZE):
                C[i][j] +=  A[i][k] * B[k][j]


print() # Lasciamo uno spazio per leggibilità


product(M1,M2,C)


for i in range(SIZE):
    print (C[i])
```

8

8. Il sistema ICD

I.C.D.
International Classification of Diseases

L'ICD è un sistema di classificazione sviluppato dall'Organizzazione Mondiale della Sanità (OMS).

- Serve per classificare le malattie, i disturbi, i traumatismi e le cause di morte in modo standardizzato, consentendo la raccolta e l'analisi dei dati sanitari a livello globale.
- L'ICD è utilizzato per scopi statistici, epidemiologici e di ricerca, facilitando il confronto dei dati sanitari tra diverse popolazioni e paesi.
- L'ICD è aggiornato periodicamente, con la decima revisione (ICD-10) attualmente in uso e la undicesima (ICD-11) in fase di sviluppo.

L'ultima versione della **ICD (International Classification of Diseases)** usata in ambito medico è:

ICD-11

- **Pubblicata dall'OMS (Organizzazione Mondiale della Sanità)** nel 2018.
- **Entrata in vigore il 1º gennaio 2022** come standard ufficiale per le statistiche sanitarie e le diagnosi cliniche a livello globale.
- È destinata a **sostituire l'ICD-10**, che era in uso dal 1990.

Caratteristiche principali dell'ICD-11:

- Completamente **digitale** e **navigabile online**.
- Maggiore dettaglio diagnostico, con oltre **55.000 codici**.

Esempio di codici ICD:

Codice Categoria Malattie e Traumatismi	Capitolo	Blocchi di categorie	Descrizione delle categorie di Malattie e traumatismi
A00	01	01	Colera
A01	01	01	Febbre tifoide e paratifo
A02	01	01	Altre infezioni da salmonella
A03	01	01	Shigellosi
A04	01	01	Altre infezioni batteriche intestinali
A05	01	01	Altre intossicazioni alimentari batteriche
A06	01	01	Amebiasi

In Italia:

L'**adozione dell'ICD-11** nei sistemi clinici e amministrativi è ancora in fase di **transizione**. Attualmente, molti sistemi sanitari (incluso quello italiano) **utilizzano ancora l'ICD-10 o addirittura l'ICD 9**, spesso in versioni adattate come l'**ICD-10-CM** (Clinical Modification) o l'**ICD-9-CM** per alcune applicazioni come la codifica delle SDO (Schede di Dimissione Ospedaliera).

S.D.O.

La **Scheda di Dimissione Ospedaliera** è un documento sanitario **obbligatorio** in Italia, compilato alla **dimissione di ogni paziente ricoverato** in una struttura ospedaliera. Serve a:

- Registrare informazioni cliniche, amministrative e diagnostiche del ricovero.
- Consentire **analisi statistiche**, epidemiologiche e per **rimborso** da parte del Servizio Sanitario Nazionale.

Relazione tra SDO e ICD:

- **Diagnosi e procedure codificate con ICD:**
    - Le **diagnosi** principali e secondarie nella SDO vengono codificate usando **codici ICD**.
    - Le **procedure/interventi chirurgici** usano anch'essi codici derivati (in Italia: **ICD-9-CM**, modificata).
- **Versione ICD usata nella SDO italiana:**
    - Ad oggi, in Italia, nella compilazione della SDO, la ICD più usata **è ancora l'ICD-9-CM** (Clinical Modification), versione americana del 1979, aggiornata localmente fino a poco tempo fa.

        - **Nonostante l'ICD-10 e l'ICD-11 esistano**, l'Italia non ha ancora adottato pienamente queste versioni nella SDO per motivi di compatibilità, sistemi informativi e transizione normativa.
- **Transizione futura:**
    - L'Italia sta **valutando il passaggio a ICD-11**, ma è un processo complesso che richiede:
        - aggiornamento dei software sanitari;
        - formazione del personale;
        - revisione dei DRG (Diagnosis Related Groups), cioè i gruppi di tariffazione collegati alle diagnosi SDO.
- Migliorata integrazione con i sistemi informatici ospedalieri e di sanità pubblica.

Ruolo del medico digitale in relazione al sistema ICD e SDO

Un laureato in **"Medicina e Chirurgia e Tecnologie Digitali"** ha **enormi potenzialità** in Italia, specialmente in relazione a sistemi come la **SDO** e le **classificazioni ICD**, perché rappresenta **l'anello mancante** tra **clinica, informatica e gestione sanitaria**.

Ecco cosa può fare di concreto e utile, in questo specifico ambito:

1. Migliorare la qualità e l'accuratezza delle SDO

- **Analisi clinica + competenza digitale** = possibilità di:
    - individuare **errori o incongruenze** nei codici ICD inseriti nella SDO;
    - migliorare la **completezza dei dati**;
    - aiutare gli ospedali a evitare **perdite economiche** causate da codifiche sbagliate che abbassano il DRG e quindi il rimborso

2. Supportare la transizione da ICD-9-CM a ICD-11

- Pochi professionisti in Italia comprendono **sia la clinica sia la struttura informatica della codifica ICD-11**.
- Un medico con competenze digitali può:
    - collaborare con Ministero e Regioni per **progettare e testare l'adozione dell'ICD-11**;
    - contribuire all'**aggiornamento dei software** ospedalieri;
    - formare il personale sanitario su **ICD-11 e nuove SDO digitali**

3. Analisi dei dati sanitari (big data sanitari)

- La SDO alimenta i **database nazionali di ricovero**.
- Un medico digitale può usare questi dati per:
    - identificare **pattern di patologie** o **errori clinici sistemici**;
    - fare **ricerca epidemiologica**;
    - costruire modelli predittivi (es. rischio di riammissione).

4. Ottimizzazione del rimborso sanitario (DRG: Diagnosis Related Groups)

- E; il sistema dottato in Italia per i rimborsi sanitari
- I **DRG** si basano sui codici ICD presenti nelle SDO.
- Un medico digitale può:
    - aiutare i reparti a **codificare in modo più preciso**;
    - massimizzare il **rimborso ottenuto dall'ASL o dallo Stato**;
    - evitare **sanzioni per errori di codifica**.

5. Sviluppo o miglioramento dei software sanitari

- Collaborare con aziende che sviluppano **software di gestione clinica o SDO**, fornendo:
    - test clinico;
    - miglioramento dell'usabilità;
    - integrazione con altre basi dati mediche internazionali

6. Ruolo ponte tra sanitari, ingegneri e amministrativi

- Essere il referente interno per progetti di:
    - **digitalizzazione ospedaliera**;
    - **telemedicina**;
    - **cartella clinica elettronica**;
    - **interoperabilità dei dati sanitari**.

Esempio applicativo in python per estrazione dati ICD da referto medico

```
import re

# Dizionario ICD semplificato (in realtà si usano database o SNOMED
CT/ICD mappings)
icd_dict = {
    "polmonite batterica": "J15.9",
    "diabete mellito tipo 2": "E11.9",
    "scompenso cardiaco": "I50.9"
}

# Funzione per cercare codici ICD nel testo
def estrai_diagnosi_icd(testo, dizionario_icd):
    diagnosi_estratte = []
    for termine, codice in dizionario_icd.items():
        if re.search(termine, testo, re.IGNORECASE):
            diagnosi_estratte.append((termine, codice))
    return diagnosi_estratte

# Esempio di referto
referto = """Il paziente è stato ricoverato per polmonite batterica
acuta.
Presenta anche diabete mellito tipo 2 ben controllato con metformina.
Nessun segno di scompenso cardiaco."""

# Estrazione
diagnosi_codificate = estrai_diagnosi_icd(referto, icd_dict)

# Output strutturato
for termine, codice in diagnosi_codificate:
    print(f"Diagnosi: {termine} → Codice ICD: {codice}")
```

OUTPUT

```
Diagnosi: polmonite batterica → Codice ICD: J15.9
Diagnosi: diabete mellito tipo 2 → Codice ICD: E11.9
```

Applicazioni reali:

- **Pre-codifica automatica della SDO**, da far validare al medico.
- Analisi epidemiologiche su **grandi quantità di referti**.
- Estrazione strutturata per **machine learning** (es. predire rischio di complicanze).