

# Guida di base alla programmazione

*di Bruno Migliaretti*



# Sommario

- 5 LA RAPPRESENTAZIONE DIGITALE DELLE INFORMAZIONI  
Come le informazioni vengono codificate per poter essere elaborate da un calcolatore elettronico.
  - 17 LA COMPRESSIONE DEI DATI  
Ridondanza delle informazioni e tecniche di compressione
  - 29 LA PROGRAMMAZIONE  
Linguaggi e tipologie d'applicazione
  - 35 ALGORITMI  
Definizione di un algoritmo, sua rappresentazione ed esecuzione
  - 43 IMPARIAMO AD INDENTARE  
Come scrivere un codice sorgente chiaro e interpretabile da altri
  - 45 INTRODUZIONE ALLA LOGICA  
L'algebra di Boole
  - 51 GLI ELEMENTI DEL LINGUAGGIO  
Parole chiave, operatori, costanti
  - 59 VARIABILI E TIPI DI DATI  
Cosa sono e come possiamo utilizzarle
  - 65 LA PROGRAMMAZIONE CONDIZIONALE  
I costrutti per creare delle condizioni all'interno del programma: IF e ELSE IF
-

## 69 LA PROGRAMMAZIONE ITERATIVA

La programmazione iterativa: WHILE, DO e FOR

## 75 FUNZIONI E METODI

Semplifichiamo il nostro lavoro di progettazione con le funzioni

## 89 LE CLASSI

I vantaggi della programmazione orientata agli oggetti.

## 93 OBJECT, ARRAY E DATE

Esempi di come funzionano alcune classi predefinite.

# Rappresentazione digitale delle informazioni

## Definizione di informazione

In termini di economia del linguaggio possiamo affermare che informazione è tutto ciò che riduce incertezza. Esempi: una foto riduce l'incertezza di chi la osserva riguardo alla scena raffigurata, l'espressione del volto di un interlocutore riduce l'incertezza sul suo stato d'animo, un fruscio riduce l'incertezza sul luogo in cui cercare un fagiano, ogni parola riduce l'incertezza sul significato di una frase.

È una definizione limitata (in termini di antropologia culturale sappiamo che non sempre ciò che aggiunge sapere sottrae incertezza, anzi...) ma che ci aiuta ad introdurre la definizione successiva: la definizione di dato.

Per dato intendiamo un'informazione codificata. Quando l'informazione viene codificata in un linguaggio convenzionale per poter essere rappresentata, scambiata, memorizzata ed elaborata otteniamo un dato.

La codifica delle informazioni può rendere omogenea la rappresentazione e l'elaborazione di informazioni di natura diversa (suoni, immagini, testi, numeri) e renderli disponibili per l'elaborazione automatica.

Informatica significa informazione automatica. Più in particolare diciamo che l'informatica è quella disciplina che studia l'elaborazione automatica delle informazioni. Condizione prima perché le informazioni possano essere elaborate e che vengano codificate in un linguaggio convenzionale che l'informatica possa elaborare.

Ora vediamo di comprendere meglio il concetto di automazione. Per automatico si intende tutto ciò che compie un compito prestabilito senza l'intervento umano. Quindi quando si parla di *automazione* si fa riferimento alla realizzazione di strumenti per la soluzione di problemi o l'esecuzione di processi in maniera indipendente dall'attività umana.

---

In termini di economia dei processi un qualche processo viene automatizzato quando il numero di volte che esso deve essere eseguito è sufficientemente grande da rendere conveniente la progettazione e la costruzione di un sistema automatico che lo risolva. Gli stessi problemi che vengono trattati in maniera automatica sono stati originariamente trattati dall'uomo.

Alcuni esempi

<b>Problema originale (non ripetitivo)</b>	<b>Problema sistematico e ripetitivo</b>
<i>Soluzione manuale</i>	<i>Soluzione automatica</i>
Determinare il diametro di una pallina	Classificare un insieme di palline in base al diametro
<i>Calibro</i>	<i>Griglie forate</i>
Scrivere una lettera	Riprodurre un testo in migliaia di copie
<i>Carta e penna</i>	<i>Stampa tipografica</i>
Regolare il traffico in caso di emergenza	Regolare il traffico ad ogni incrocio
<i>Vigile urbano</i>	<i>Semaforo</i>
Riempire un contenitore d'acqua	Riempire ripetutamente un contenitore con la stessa quantità d'acqua
<i>Rubinetto manuale</i>	<i>Rubinetto a galleggiante</i>

## Che cosa è un computer

Un elaboratore elettronico è una macchina (elettronica) programmabile per l'elaborazione automatica di dati. Un elaboratore è costituito da una parte così detta Hardware (componenti tangibili e permanenti di un computer). E da una parte software (insieme dei programmi, componenti intangibili e temporanei, che controllano il funzionamento dell'hardware).

L'hardware è in grado di eseguire un insieme finito di operazioni elementari e di interpretare istruzioni in un linguaggio predefinito. Il software specifica sequenze di istruzioni da eseguire per arrivare alla soluzione di un problema.

Un computer è programmabile. È in grado, cioè, di svolgere compiti diversi in base ad un programma. Un elaboratore può essere utilizzato ripetutamente per risolvere problemi diversi o istanze diverse dello stesso problema. In molti casi, l'automazione di un processo è conveniente solo se

si può realizzare con elaboratori programmabili già progettati e sviluppati per altre applicazioni. In tal caso solo il programma deve essere riprogettato. Il processo di codifica consente di utilizzare lo stesso elaboratore per manipolare informazioni diverse. Ogni informazione può (a meno di un'approssimazione) essere rappresentata come una sequenza di due soli simboli (di solito 0 e 1) che nel campo informatico vengono chiamati BIT.

La parola BIT nasce dalla contrazione delle parole inglesi Binary Digit e letteralmente BIT significa "cifra binaria". Cioè una cifra che può assumere solo due valori (di solito si parla di 0 e 1). La rappresentazione fisica di un bit si può ottenere da un qualunque sistema in grado di trovarsi in due stati diversi mutuamente esclusivi. Ad esempio un bit può essere rappresentato da un interruttore che può trovarsi nei due stati acceso/spento. L'interruttore infatti si trova alla base del funzionamento dei circuiti logici che costituiscono le unità di elaborazione dei microprocessori.

Allo stesso modo un bit può essere rappresentato da un condensatore che si trova nei due stati carico/scarico. Tutte le memorie principali così dette dinamiche (come la RAM) si basano su questo principio, memorizzando i singoli bit in piccolissimi condensatori in modo tale da rappresentare un 1 se il condensatore è carico e uno 0 se questo è scarico.

D'altra parte, nelle memorie di massa non volatili come i dischi fissi, i bit vengono memorizzati attraverso particelle magnetiche orientabili. In questo modo per rappresentare un bit di valore 1 si orienta una particella in una direzione e per rappresentare uno 0 si orienta nel verso opposto.

La codifica delle informazioni ci consente, cioè, di utilizzare determinati supporti fisici (hardware) per rappresentarle.

## Codifica delle informazioni

La codifica delle informazioni in termini di bit è l'operazione che consente di utilizzare i calcolatori elettronici per elaborare dati di qualsiasi natura. Abbiamo già detto che un bit può rappresentare solamente due diverse informazioni (1 o 0, acceso o spento, ecc.). Per codificare insieme con un numero maggiore di informazioni/stati usiamo stringhe di bit di lunghezza variabile. Una stringa è una sequenza di  $n$  bit e viene anche detta parola dall'inglese word. Con un numero  $n$  bit si possono formare  $2^n$  configurazioni diverse e quindi si possono rappresentare  $2^n$  informazioni diverse. La codifica è una convenzione che associa un significato ad ogni configurazione di bit.

I calcolatori sono oggetti finiti che elaborano e memorizzano un numero finito di bit. Un numero finito di bit permette di codificare un numero finito di informazioni diverse. Se l'insieme delle informazioni da rappresentare è superiore alle combinazioni di BIT che ho a disposizione non avrò una

---

rappresentazione esatta delle informazioni ma una rappresentazione approssimata.

Un insieme di informazione può essere limitato o illimitato (infinito) e può essere discreto a continuo. Facciamo alcuni esempi.

L'insieme dei caratteri di una lingua è limitato e discreto in quanto composto da un numero definito di segni.

L'insieme dei numeri interi è infinito e discreto: la sequenza dei numeri è infinita, ma, dati due valori qualsiasi, il numero degli elementi compresi tra i due valori è sempre finito.

L'insieme dei numeri reali è invece infinito e continuo: la sequenza dei numeri reali è infinita e il numero di elementi compresi tra due elementi qualsiasi della sequenza anche molto vicini tra loro è sempre infinito.

L'insieme dei colori è limitato e continuo: la sensibilità del mio occhi limita la sequenza dei colori dal rosso al violetto, ma tra due elementi della sequenza potrò sempre immaginare un numero infinito di sfumature.

La strategia adottata nella codifica cambia a seconda delle caratteristiche dell'insieme di informazioni da rappresentare.

- Normalmente gli insiemi finiti e discreti (come i caratteri di una lingua) vengono rappresentati esattamente.
- Gli insiemi illimitati vengono limitati. Nella convenzione che definisce la codifica si stabilisce che venga rappresentato un sottoinsieme di informazioni la cui dimensione dipende dal numero di bit che si dedicano alla codifica (gli elementi del sottoinsieme sono rappresentati esattamente, gli altri non sono rappresentati).
- Gli insiemi continui vengono partizionati. Normalmente l'insieme viene suddiviso in intervalli di uguale ampiezza e viene codificato come se fosse un insieme discreto.

Gli insiemi illimitati e continui saranno sia limitati che partizionati.

## Codifica del Testo

Un testo è una sequenza di caratteri alfabetici, separatori e caratteri speciali come punteggiatura, accentuazioni, ecc. La codifica ASCII (American Standard Code for Information Interchange) prevede l'uso di 128 caratteri diversi. Ogni carattere è associato ad una diversa configurazione di 7 bit.

La codifica ASCII si è piuttosto limitata. Modellata sulle lingua inglese non comprende le lettere con modificatori quali accenti, dieresi, tilde ecc. Per

---



queste limitazioni sono nate estensioni non standard del codice ASCII che utilizzano 8 bit (1 byte) per carattere, portando a 256 il numero di caratteri disponibili. Si sono diffuse quindi diverse tabelle di caratteri (specializzate per gruppi di lingue) che hanno in comune i primi 128 caratteri (ASCII standard) e usano i rimanenti per offrire alle lingue di riferimento tutti i caratteri necessari. Questo processo ha reso la codifica dei caratteri relativa. Il codice di un carattere non è univoco ma dipende dalla tabella che viene impiegata. Ad esempio in un computer che esegue Microsoft Windows utilizzando la tabella codici predefinita 1252 e in un computer Apple Macintosh che utilizza la tabella codici Macintosh Roman, la posizione 232 di queste tabelle dei caratteri corrisponde rispettivamente alla lettera **è** nel sistema Windows ed alla lettera **Ē** in quello Macintosh.

Con l'entrata delle lingue asiatiche nel mondo dei personal computer è nata l'esigenza di aumentare in maniera sensibile il numero dei segni da codificare. Nasce quindi due progetti col fine di costruire un'unica tabella universale dei caratteri, il progetto ISO 10646 e il progetto Unicode.

Lo standard ISO10646/Unicode si basa su una codifica a 32 bit che consente oltre due miliardi di possibili caratteri, numero che dovrebbe essere decisamente sufficiente per realizzare una tavola di codifica dei caratteri veramente universale.

L'implementazione informatica di Unicode tuttavia comporta dei problemi, in particolar modo per quanto riguarda la crescita della dimensione dei files, per tal motivo sono stati creati dei charset detti UTF, *Universal Character Set Transformation Format*.

UTF usa una tecnica di bit-shifting (spostamento dei bit) per codificare i caratteri Unicode. In sostanza l'UTF usa 7 bit per carattere per codificare i primi 127 caratteri corrispondenti all'ASCII standard, e attiva l'ottavo bit solo quando serve la codifica Unicode. Ogni carattere Unicode viene quindi codificato con un numero variabile di byte che va da 1 a 3. Il vantaggio è che il testo è visualizzabile con un normale editor di testi, anche se naturalmente i caratteri diversi dall'ASCII standard potrebbero apparire non correttamente. Per i testi comuni utilizzati in occidente consente di mantenere più compatte le dimensioni dei files rispetto ad UCS in quanto la maggior parte dei caratteri sono ASCII. Inoltre, l'UTF-8 attraversa indenne, come se fosse un normale testo, anche i sistemi e i programmi che non supportano Unicode. Per questi motivi l'UTF è molto usato nella posta Internet ed in alcuni sistemi operativi.

Tra le codifiche UTF la più comune è la UTF-8, che tra l'altro è la codifica di default di XML.

Indipendentemente dalla codifica usata un testo è rappresentato dalla sequenza di byte associati ai caratteri che lo compongono, nell'ordine in cui essi compaiono. Un testo di 1000 caratteri richiede 1000 byte (1 Kb) se

---

viene utilizzata una tabella di testo ANSI, 4000 byte (4 Kb) se viene utilizzata la codifica Unicode e un numero variabile di byte, che dipende dai caratteri codificati se si usa una codifica UTF.

## Codifica dei numeri

Per la rappresentazione dei numeri è particolarmente diffusa la codifica binaria, che fornisce una diretta trasposizione in un alfabeto a due valori della notazione decimale da noi comunemente utilizzata. Il numero di simboli dell'alfabeto è detto base di numerazione. La codifica binaria e quella decimale utilizzano basi diverse (2 e 10, rispettivamente) ma sono entrambe notazioni posizionali, in quanto permettono di rappresentare ogni numero come sequenza di cifre del tipo:

$$c_n \ c_{n-1} \ \dots \ c_1 \ c_0 \ c_{-1} \ c_{-2} \ \dots \ c_{-m}$$

Il valore ( $v$ ) del numero rappresentato è la somma pesata delle cifre, dove il peso di ogni cifra dipende dalla base di numerazione ( $B$ ) e dalla posizione della cifra:

$$v = c_n B^n + c_{n-1} B^{n-1} + \dots + c_1 B + c_0 + c_{-1} B^{-1} + c_{-2} B^{-2} + \dots + c_{-m} B^{-m}$$

Es:  $(101.1)(\text{base } 2) = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} = (5.5)(\text{base } 10)$ .

### Numeri interi.

I numeri interi sono un insieme discreto illimitato. Per poter essere codificati devono essere limitati. In genere ci si restringe ad un sottoinsieme simmetrico rispetto allo 0. Avendo a disposizione  $n$  bit, se ne usa uno per rappresentare il segno e i restanti  $n-1$  per rappresentare il modulo. Se  $a = n-1$  (il numero dei bit a disposizione meno uno) il massimo numero rappresentabile è (in modulo)  $2^{a-1}$ . Se il risultato di un'operazione superiore a  $2^{a-1}$  o inferiore a  $-2^{a-1}$  non può essere codificato e il calcolatore restituisce un messaggio di overflow.

### Numeri reali.

I numeri reali sono un insieme continuo e illimitato. Per poterli rappresentare occorre limitarli (in modo simmetrico rispetto allo 0) e partizionarli. Esistono due tipi di rappresentazione dei numeri reali: rappresentazione in virgola fissa e rappresentazione in virgola mobile o *floating-point*.

#### Rappresentazione in virgola fissa.

Nella rappresentazione a virgola fissa un bit viene serve a indicare il segno (+ o -) un certo numero di bit (diciamo  $a$ ) vengono dedicati a rappresentare le cifra prima della virgola e i rimanenti (diciamo  $b$ ) le cifre dopo la virgola. Più  $a$  sarà grande più potremo rappresentare numeri positivi grandi e

numeri negativi piccoli, più  $b$  sarà grande più aumenterà la precisione. Il difetto della rappresentazione dei numeri reali in virgola fissa è la rigidità.

Rappresentazione in virgola mobile (floating-point).  
Il numero da rappresentare viene espresso nella forma

$$S \ 0.M \ B^{seE}$$

dove  $s$  rappresenta il segno,  $M$  la mantissa,  $E$  l'esponente, se il segno dell'esponente e  $B$  la base di numerazione. Non tutti i numeri hanno rappresentazione univoca. Questo comporta perdita di efficienza e difficoltà di confronto. Quando esistono rappresentazioni equivalenti dello stesso numero, tra queste si definisce normale quella con l'esponente più piccolo. Da una rappresentazione non normale a quella normale equivalente si passa traslando verso sinistra le cifre della mantissa e sottraendo 1 all'esponente.

## Codifica delle immagini

Le immagini sono informazioni continue in tre dimensioni: due spaziali ed una colorimetrica. Per codificarle occorre operare tre discretizzazioni. Le due discretizzazioni spaziali riducono l'immagine ad una matrice di punti colorati, detti pixel. La terza discretizzazione limita l'insieme di colori che ogni pixel può assumere.

### Immagini bitmap

Il modo più elementare per rappresentare un'immagine consiste nel rappresentare il colore di ogni pixel. Il numero di bit necessari a rappresentare ogni pixel dipende dal numero di diversi colori disponibili. Un byte consente di rappresentare 256 colori. Un'immagine a 256 colori è una sequenza di byte.

Un'immagine di 100X100 pixel a 256 colori richiede 10000 byte (10 Kb) per essere rappresentata. Un'immagine a due soli colori (ad esempio bianco e nero) richiede un solo bit per pixel. Le immagini codificate secondo questa tecnica vengono dette anche immagini raster o immagini bitmap. Questo tipo di immagini vengono solitamente usate per fotografie, scansioni e in genere per tutte le immagini che prevedono sottili gradazioni nei colori e nelle forme.

Le immagini raster sono caratterizzate da tre informazioni principali che ne consentono la corretta interpretazione: dimensioni, numero dei colori, risoluzione.

Le dimensioni consentono di ricostruire correttamente l'immagine collocando i pixel al loro posto.

---

Il numero dei colori è definito dal numero di bit che viene dedicato alla descrizione del colore di ogni pixel:

- immagini monocromatiche: un bit per ogni pixel (se il bit è 0 è bianco, se no nero)
- immagini a 256 colori: 8 bit (un byte) per ogni pixel. A ogni immagine è associata una tabella di 256 colori (detta tavolozza o palette) in cui sono descritti i contenuti RGB (Red, Green, Blue) di ognuno dei 256 colori usati. In altre parole i 256 sono numerati da 0 a 255 e l'elaboratore, quando decodifica l'immagine controlla a che colore corrisponde il numero che descrive ogni pixel. In questa maniera si possono avere in ogni immagine al massimo solo 256 colori diversi ma scelti
- immagini a 16 bit: vengono riservati 16 bit (2 byte) per descrivere il colore di ogni pixel. Ogni colore è descritto dalle sue componenti RGB (Red, Green, Blue). Vengono messi a disposizione 5 bit per ogni componente (l'ultimo bit non è usato). Avremo cioè colori descritti con  $2^5$  (cioè 32) livelli di Rosso, Verde e blue per un totale di 32.768 colori.
- immagini a 24 bit: vengono riservati 24 bit (3 byte) per descrivere il colore di ogni pixel. Ogni colore è descritto dalle sue componenti RGB (Red, Green, Blue). Vengono messi a disposizione 8 bit (un byte) per ogni componente. Avremo cioè colori descritti con  $2^8$  (cioè 256) livelli di Rosso, Verde e blue per un totale di 16.777.216 colori.
- immagini a 32 bit: vengono riservati 32 bit (4 byte) per descrivere il colore di ogni pixel. Come nelle immagini a 24 bit vengono messi a disposizione 8 bit (un byte) per ogni componente. Il quarto byte descrive 256 livelli di trasparenza (il cosiddetto canale alpha) che può avere il pixel.

Infine la risoluzione definisce il rapporto che c'è tra la dimensione dell'immagine in pixel e la dimensione reale che l'immagine ha quando viene resa da un dispositivo (stampante, fax, schermo, ecc). La risoluzione di misura normalmente in punti (pixel) per pollice o in punti per centimetro.

## Immagini vettoriali

La grafica vettoriale scompone l'immagine in gruppi logici di componenti (linee, cerchi, rettangoli, ecc.) e memorizza le forme in termini di coordinate e colori dei vari elementi geometrici che le compongono. In definitiva un file che memorizza un'immagine di tipo vettoriale è costituito da un'insieme di coordinate e dalle informazioni sul colore di ciascuna forma. Nel momento della visualizzazione i punti descritti dalle coordinate vengono disegnati e

---

colorati a dovere. La grafica vettoriale viene comunemente usata nel disegno animato, nella grafica lineare e negli strumenti CAD.

Un grande vantaggio delle immagini vettoriali rispetto a quelle raster/bitmap sta nella minor dimensione del file che le memorizza e nella capacità di essere variate di dimensioni senza subire alcuna distorsione. Le immagini bitmap, invece, se ridimensionate rispetto alle dimensioni originali di acquisizione, hanno la tendenza a perdere di risoluzione risultando distorte o sfocate.

Filmati video (immagini in movimento). I filmati video possono essere considerati sequenze di fotogrammi. Abbiamo già visto come codificare immagini, e quindi sappiamo come codificare ogni fotogramma. Il numero di fotogrammi nell'unità di tempo determina la qualità del movimento. E' stato sperimentato che l'occhio umano non percepisce discontinuità nel movimento al di sotto del venticinquesimo di secondo. Pertanto, una frequenza di riproduzione dei fotogrammi superiore a 25Hz sarebbe superflua se lo scopo fosse la riproduzione di un video. D'altro canto potrebbe non esserlo se servisse a documentare un esperimento di brevissima durata o se il video dovesse essere rivisto al rallentatore.

Es: Filmato di 10 minuti, con risoluzione di 100x100 pixel a 256 colori: dimensione complessiva di  $600 \times 25 \times 100 \times 100 \times 8 = 1.2\text{Gbit}$ .

## Codifica di segnali analogici

Un segnale analogico è una grandezza fisica che varia nel tempo e che può assumere un insieme continuo (e quindi infinito) di valori. Un segnale digitale invece può assumere solo un insieme discreto (e generalmente finito) di valori. Un segnale analogico ha anche la proprietà di essere tempo-continuo: cioè il suo valore è significativo (e può variare) in qualsiasi istante di tempo.

Diversamente, un segnale digitale viene anche detto segnale tempo-discreto: poiché il suo valore ha interesse solo in istanti di tempo prestabiliti, generalmente equidistanziati, mentre non ha alcun valore tra un istante ed un altro.

La rappresentazione più naturale di un segnale  $s$  è la sequenza dei suoi valori istantanei  $s(t)$ . Ci poniamo prima il problema di rappresentare un valore istantaneo, e poi quello di rappresentarne la sequenza.

Il valore istantaneo di un segnale può essere rappresentato da un numero, e come tale può essere codificato utilizzando notazione in virgola fissa o in virgola mobile. Tali codifiche inducono limitazioni e partizionamenti sullo spazio dei possibili valori che il segnale può assumere (quantizzazione del segnale) .

---

Una volta codificato il valore istantaneo del segnale si ottiene la sequenza temporale ripetendo la codifica ad intervalli prestabiliti. In generale, se il segnale da codificare era tempodiscreto, la scelta naturale è quella di rappresentare il valore del segnale in tutti e soli gli istanti in cui esso è significativo. Per la codifica dei segnali analogici, che come abbiamo detto sono tempocontinui occorre operare una discretizzazione dell' intervallo temporale per ottenere una rappresentazione finita. L'operazione di discretizzazione più comune è il campionamento, che consiste nell'osservare il segnale tempo-continuo ad istanti di tempo prefissati (generalmente equidistanziati) in modo da ottenere un segnale tempodiscreto che assume lo stesso valore di quello originale nei punti di campionamento.

In definitiva, le operazioni di campionamento e quantizzazione trasformano un segnale analogico tempocontinuo in un segnale digitale tempodiscreto.

## Codifica del suono

Il suono è un segnale analogico tempo-continuo. Ci sono diversi modi per descrivere la natura di tale segnale. Per fissare le idee pensiamo che il suono prodotto da un altoparlante è prodotto dalla vibrazione di una membrana. Descrivendo la posizione della membrana nel tempo (e quindi il suo spostamento) a tutti gli effetti descriviamo il suono. Assumiamo allora che il segnale analogico  $s(t)$  esprima la posizione della membrana (rispetto ad un riferimento fissato) all' istante di tempo  $t$ . Tale segnale deve essere campionato e discretizzato per poter essere codificato in termini di bit. Per valutare la qualità della codifica possiamo pensare di usare il segnale quantizzato e tempo-discreto (quello cioè effettivamente rappresentato dalla nostra codifica) per muovere la membrana di un altoparlante. Se il campionamento è troppo rado e vengono usati pochi bit per codificare ogni valore istantaneo, la perdita di informazione degrada la qualità del suono (il suono riprodotto è sensibilmente diverso da quello originale).

Per quanto riguarda il campionamento, è dimostrato che il suono percepibile dall'orecchio umano viene riprodotto fedelmente se la frequenza di campionamento (il numero di campioni in un secondo) è non inferiore a 30KHz. Per rendere intelligibile il parlato è sufficiente una frequenza di campionamento di 8KHz. La quantizzazione introduce comunque una distorsione. Questa tecnica di codifica del suono è quella comunemente usata nella codifica di tipo wave che troviamo nei file .wav. Questa codifica tenta di ricostruire fedelmente la forma d'onda del suono campionandola adeguatamente. Il file che si ottiene è normalmente di grandi dimensioni.

Es: Telefono. La comunicazione telefonica deve riprodurre la voce umana ai due estremi del collegamento garantendo la comprensione del parlato. A tal fine lo standard prevede un campionamento a 8KHz ed una quantizzazione a 256 livelli (codificati con 8 bit). Il segnale digitale che ne risulta usa 64Kbit

---

per codificare ogni secondo di conversazione. 10 minuti di conversazione richiedono 38.4Mbit.

La codifica di tipo MIDI, invece ha tra i formati audio lo stesso ruolo che il formato vettoriale ha tra i formati grafici. Infatti, il formato MIDI acronimo di Musical Instrument Digital Interface, non è una registrazione sonora ma contiene dei comandi da impartire ad un sintetizzatore sonoro (software o hardware) che riproduce il suono di uno strumento musicale. I file MIDI per questo motivo sono molto piccoli ed hanno il formato del tipo:

<b>suona la nota x per un tempo y con lo strumento z.</b>
---

La codifica MIDI, quindi riproduce un suono in maniera sintetica. Le nuove suonerie polifoniche dei telefoni cellulari di ultima generazione si basano su una codifica di tipo MIDI.

---





## La compressione dei dati

Spesso risulta utile ridurre le dimensioni dei dati su si sta lavorando in modo da renderne più agevole l'archiviazione e la trasmissione. Una condizione tipica in cui viene applicata la compressione sui dati è la ridondanza. In poche parole, dei dati che contengono le stesse informazioni in qualche modo ripetute (ridondanza) sono facilmente comprimibili. Ad esempio, quando ci si trova di fronte a un testo scritto si incorre in esempi di ridondanza lampanti. Si consideri, per esempio, l'uso che la lingua italiana fa della lettera "q" che è sempre seguita da una "u": nessuno avrebbe difficoltà nel capire il significato della parola "q\*adro'", nonostante essa non sia scritta correttamente. Quindi, una prima tecnica di compressione della lingua italiana potrebbe essere basata sull'omissione del carattere "u" tutte le volte che questo è preceduto da un carattere "q".

La presenza di ridondanza in una lingua, in un testo scritto, o in una qualunque stringa di simboli non è però nefasta, anzi, essa svolge il compito fondamentale di rendere robusto il messaggio contenuto in quella stringa. Per robusto si intende facilmente comprensibile e non incline ad essere male interpretato anche nel caso in cui il messaggio venga trasmesso in modo solo parziale. Errori di trasmissione su codici ridondanti sono molto più facilmente correggibili.

D'altra parte, un testo che sia stato compresso al massimo è molto fragile, in quanto, per definizione di massima compressione, ciascun simbolo sarà portatore di informazione, e quindi una sua alterazione porterà ad una perdita irrimediabile di quella stessa informazione.

In definitiva, l'uso di codifiche ridondanti può avere due motivazioni: la flessibilità e l'affidabilità.

- La flessibilità indica la possibilità di utilizzare la stessa codifica in situazioni diverse. Es: il cosiddetto "baco del millennio" (millennium bug) che costa al mondo migliaia di miliardi è dovuto alla scarsa lungimiranza con cui i produttori di software hanno deciso di utilizzare due sole cifre decimali per rappresentare l'anno nelle date, dando per scontate (e quindi prive di contenuto informativo) le prime 2, fino ad ora sempre uguali a 19.

- L'affidabilità deriva dalla capacità di alcune codifiche ridondanti di rivelare o correggere errori. Un codice non ridondante associa univocamente una configurazione ad un significato. Un errore nella configurazione comporta un errore di interpretazione del significato. Una codifica ridondante a rivelazione d'errore associa significati utili solo ad un sottoinsieme delle configurazioni possibili. Se un errore trasforma una configurazione significativa in una non significativa, chi la interpreta riconosce la presenza dell'errore. Una codifica ridondante a correzione d'errore associa molte configurazioni allo stesso significato. Se un errore trasforma una configurazione in una a cui è associato lo stesso significato, l'interpretazione resta corretta.

Codifiche ridondanti si prestano a compressione, come dimostrano gli esempi precedenti, ma esistono anche tecniche di compressione che agiscono su dati irridondanti sfruttando proprietà tipiche dell'informazione da rappresentare.

## Tecniche generiche di compressione

Per quanto riguarda le tecniche di compressione una prima distinzione che bisogna fare è tra compressione lossy e lossless. La prima tecnica effettua una compressione dei dati definita " con perdita d'informazione" o anche distruttiva in quanto, una volta applicata questa tecnica, non è più possibile ricostruire in maniera esatta i dati di partenza attraverso il processo di decompressione.

In definitiva, c'è stata, una perdita irrimediabile di informazione.

Le tecniche di tipo lossy sono legate a specifiche codifiche delle informazioni (immagini, suoni, filmati, ecc.) e sfruttano le loro specifiche caratteristiche. In questi casi perdita di informazione significa perdita di qualità.

Questa tecnica è molto utilizzata nel campo della compressione dei formati multimediali soprattutto laddove è spesso inutile ricostruire, come nel caso delle fotografie, l'informazione iniziale con una risoluzione maggiore di quella che l'occhio umano può apprezzare o di quella che uno specifico mezzo (ad esempio il video) può restituire.

Le tecniche di compressione di tipo lossless permettono invece di recuperare interamente l'informazione contenuta nel documento codificato prima della sua compressione, ma la loro efficienza di compressione è minore. Queste tecniche sono comunque le uniche utilizzabili in quei casi in cui non è possibile accettare neppure la minima perdita delle informazioni. Ad esempio la tecnica di compressione che sta alla base dei compressori tipo Winzip o Winrar è una tecnica di tipo lossless in quanto non sarebbe accettabile perdere dell'informazione comprimendo ad esempio un file di installazione di un programma o un file di testo.

---

## Tecniche di tipo lossless

### Compressione run-length

La compressione di tipo run-length fa parte delle tecniche di compressione di tipo lossless cioè senza perdita di dati. Essa si basa su un algoritmo denominato di *Run Length Encoding* (RLE) che è uno dei più semplici algoritmi di compressione mai progettati. Si basa sul fatto che nei dati da comprimere esistono sequenze, definite run, che si ripetono costantemente. Una volta individuate le sequenze ripetute, vengono sostituite da un unico simbolo e dal numero delle ripetizioni presenti.

Ad esempio data la stringa di bit 01110000 l'algoritmo RLE la comprimerebbe codificandola in 03\*14\*0 che sta ad indicare "0 tre volte 1 quattro volte 0" .

Esistono numerose varianti di RLE le cui principali differenze consistono nella lunghezza minima da attribuire ad un run. Nell' esempio appena descritto abbiamo usato una lunghezza del run pari a uno cioè si prendeva in considerazione un carattere alla volta e si cercava se questo veniva

ripetuto consecutivamente. Allo stesso modo si possono prendere in considerazione gruppi di simboli e cercare se l' intero gruppo viene ripetuto all' interno della stringa.

### Compressione a codifica differenziale

In alcuni casi, le informazioni sono costituite da blocchi di dati, ognuno dei quali differisce leggermente dal precedente come, ad esempio, i fotogrammi successivi di un filmato. In questo caso sono utili le tecniche di compressione che utilizzano la codifica differenziale. L'approccio di queste tecniche è quello di memorizzare non il blocco stesso ma le sue differenze rispetto al precedente. È evidente che in questo caso se si dovesse corrompere un blocco compresso durante la trasmissione, risulterebbe compromessa la ricostruzione/decompressione di tutti i blocchi successivi.

### Codifiche basate su dizionari

Il termine dizionario si riferisce all' insieme di elementi di base sui quali viene ricostruito il messaggio compresso. In pratica viene utilizzato un insieme di simboli (dizionario) per codificare un messaggio. I simboli del dizionario rappresentano particolari sequenze di bit e durante la compressione, ad ogni sequenza riconosciuta viene sostituito il simbolo corrispondente. La particolarità di queste tecniche sta nel fatto che il dizionario viene creato dinamicamente durante il processo di compressione. Tale tecnica sta alla base dell' algoritmo di compressione Lempel-Ziv che troviamo nella maggior parte dei tools di compressione quali WinZip.

---

# Compressione delle immagini

## Formato GIF

La sigla GIF è acronimo di Graphic Interchange Format. Questo tipo di compressione rientra nelle tecniche di tipo lossless, cioè senza perdita di dati. La caratteristica del formato GIF è che esso può esportare solo immagini che contengono al massimo 256 colori. Se l'originale contiene un numero più elevato di colori quindi, è necessario effettuare una riduzione e la perdita di qualità sarà significativa. Il formato GIF usa colori a 8 bit ed è efficace per comprimere immagini vettoriali, geometriche o testo. Questo formato fu diffuso negli anni Ottanta come metodo efficiente di trasmissione delle immagini su reti di dati. All'inizio degli anni Novanta i progettisti originali del web lo adottarono per l'efficienza che offriva. Oggi la stragrande maggioranza delle immagini sul web è in questo formato ed è supportato da tutti i browser web.

Il formato GIF usa una forma di compressione LZW che mantiene inalterata la qualità dell'immagine, ovvero riduce le dimensioni del file senza pregiudicare la qualità grafica dell'immagine. Questo formato è basato sull'uso di una tavolozza di colori: anche se il singolo colore può essere uno fra milioni di sfumature, solo un certo numero di essi è disponibile (al massimo  $256=8\text{bit}$ ). I colori sono memorizzati in una 'tavolozza', una tabella che associa un numero ad un certo valore di colore.

La limitazione a 256 colori appariva ragionevole all'epoca della creazione del formato GIF perché non erano ancora diffusi dispositivi in grado di visualizzarne un numero superiore. Per disegni al tratto, fumetti, fotografie in bianco e nero sono di regola sufficienti 256 colori. Minore sarà il numero di colori presenti nell'immagine e maggiori saranno le possibilità di compressione, ovvero minori saranno le dimensioni del file in quanto sarà ridotta la dimensione della tavolozza e quindi il numero di codici per descrivere i colori.

Il formato GIF consente anche di salvare le immagini in un formato interlacciato. Il formato a interlacciamento produce una visualizzazione graduale di un'immagine in una serie di passate sempre più definite a mano a mano che i dati arrivano al browser. Ogni nuovo passo crea un'immagine più nitida fino al completamento dell'intera immagine.

Il formato GIF consente anche di definire un colore come trasparente. Nelle aree di colore contrassegnato come trasparente, verrà visualizzato il colore di sfondo. Questa proprietà viene utilizzata per le animazioni e per la sovrapposizione di più immagini.

Il formato GIF, infine prevede una serie di semplici comandi con cui si può stabilire la frequenza con cui il fotogrammi di una animazione sono visualizzati, se l'animazione è continua o viene ripetuta un numero finito di

---

volte, se le immagine visualizzate vengono sovrapposte a quelle precedenti o lo sfondo viene cancellato. In altre parole GIF è anche un formato programmabile.

Con l'evoluzione dei personal computer (maggiore risoluzione dei video, maggior numero di colori riproducibili), oggi si usano principalmente immagini a 16, 24 o 32 bit. Ciò nonostante, anche se in misura minore, il formato GIF è ancora molto utilizzato sia in quanto formato che consente animazioni programmate, sia perché, con determinate immagini risulta più efficiente del formato JPEG (oggi sicuramente il formato di compressione più utilizzato). In realtà oggi utilizzare il formato GIF significa perdita di informazione. Il processo di conversione avviene di fatto in due fasi: una prima fase (lossy) in cui l'immagine originale (normalmente a 24 o 32 bit) viene ridotta a 256 colori creando una palette ottimizzata per rappresentarla al meglio e una seconda fase (lossless) in cui all'immagine viene applicata la compressione GIF.

Molto spesso il formato GIF risulta particolarmente efficiente quando si convertono in bitmap disegni realizzati con programmi per disegno vettoriale con limitato numero di colori e uso di colori pieni.

## Formato JPEG

Un formato grafico utilizzato più frequentemente sul Web per ridurre le dimensioni dei file grafici è lo schema di compressione JPEG (*Join Photographic Expert Group*). A differenza delle immagini GIF, le immagini JPEG sono policrome (24 bit, o 16,8 milioni di colori). Questo tipo di immagini ha generato un altissimo interesse tra fotografi, artisti, progettisti grafici, specialisti della composizione di immagini mediche, storici dell'arte e altri gruppi per i quali la qualità dell'immagine è d'importanza fondamentale e per i quali non è possibile accettare compromessi sulla fedeltà dei colori tramite retinatura di un'immagine a colori a 8 bit.

Una forma più recente di JPEG, chiamata JPEG progressivo, conferisce alle immagini JPEG la stessa gradualità di visualizzazione delle immagini GIF interlacciate; al pari di queste ultime, le immagini JPEG progressive impiegano spesso un tempo maggiore per lo scaricamento sulla pagina rispetto ai JPEG standard, ma offrono al lettore un'anteprima più rapida.

La compressione JPEG utilizza una sofisticata tecnica matematica, chiamata trasformazione discreta del coseno, per produrre una scala scorrevole di compressione delle immagini. Tale tecnica si basa su di una codifica dell'immagine percettiva in cui viene distinta la luminosità dei pixel dal loro colore. Il motivo di tale distinzione è che l'occhio umano è più sensibile alle variazioni di luminosità che non a quelle di colore. Lo standard base di JPEG trae vantaggio da questo fenomeno codificando ogni componente della luminosità, ma dividendo l'immagine in blocchi di quattro pixel e registrando solo il colore medio di ogni blocco. La rappresentazione finale

---

preserva dunque i cambiamenti di luminosità, attenuando i repentini mutamenti cromatici dell'immagine originale. Il vantaggio è che ogni blocco di quattro pixel è rappresentato soltanto da 6 valori (4 di luminosità e 2 di colore) anziché 12 valori che sarebbero necessari in un sistema a 3 valori per pixel (immagini a 24 bit RGB).

È possibile scegliere il grado di compressione che si desidera applicare a un'immagine in formato JPEG, ma in questo modo si determina anche la qualità dell'immagine. Più si comprime un'immagine con la compressione JPEG, più si riduce la qualità dell'immagine stessa.

## Formato PNG

Il formato PNG (Portable Network Graphic) è stato sviluppato appositamente per il Web.

Questo formato è stato disponibile fin dal 1995 ma ha stentato ad acquisire popolarità a causa della mancanza di un supporto generalizzato da parte dei browser. Si tratta di un formato che secondo le intenzioni degli autori doveva sostituire il formato GIF. Questo formato senza perdita di informazioni comprime le immagini a 8 bit producendo file di dimensioni inferiori rispetto a GIF ma supporta anche immagini a 16 e 24 bit.

Anche se il formato PNG supporta il colore a 24 bit, la sua routine di compressione senza perdita di informazioni non è in grado di raggiungere l'efficienza del formato JPEG. Il formato PNG supporta, inoltre, le funzionalità di trasparenza e interallacciamento ma non l'animazione.

Un'utile caratteristica del formato PNG è la capacità di incorporare del testo per offrire la possibilità di eseguire ricerche sulle immagini; è infatti possibile memorizzare nel file dell'immagine una stringa che identifica l'immagine stessa. Purtroppo il formato grafico PNG non è ampiamente supportato e l'implementazione corrente delle immagini PNG in Netscape Navigator e Microsoft Internet Explorer non supporta completamente tutte le sue funzioni.

## Compressione audio

Come nel caso delle immagini si tratta di distinguere tecniche di compressione senza perdita (lossless) e con perdita (lossy).

### Compressione senza perdita

Usando un algoritmo di compressione senza perdita, dal risultato della compressione si può riottenere tutta l'informazione originaria. In questo caso la riduzione massima generalmente ottenibile, utilizzando algoritmi studiati appositamente per l'audio è all'incirca del 60%, ma solo con alcuni tipi di suono. Si possono utilizzare gli stessi algoritmi generali di

---

compressione (come LZW) ma i risultati in termine di riduzione sono inferiori.

Un buon esempio di algoritmo di compressione lossless è il FLAC (Free Lossless Audio Codec). FLAC è un diffuso codec audio libero di tipo lossless, cioè senza perdita di qualità. La compressione non rimuove informazioni dal flusso audio, ed è quindi adatto sia all'ascolto normale che per l'archiviazione. Il formato FLAC attualmente ha un buon supporto da parte di vari software audio.

Altri esempi di compressione lossless sono APE, ALF.

### Compressione con perdita

Gli studi di psicoacustica hanno permesso di accertare che l'uomo non è sensibile nello stesso modo a tutte le frequenze e che un suono ad alta intensità ne maschera uno con frequenza vicina ma intensità più bassa. Sfruttando queste ed altre considerazioni, si può pensare di eliminare l'informazione che non verrebbe comunque percepita ed ottenere quindi un buon rapporto di compressione.

In questo modo sono stati sviluppati algoritmi di compressioni specifici per l'audio ad altissima efficienza che sono in grado di ottenere riduzione della lunghezza dei file dell'ordine di 10 a 1 praticamente senza perdita di qualità.

Facciamo alcuni esempi.

#### MP3

**MP3** (o, più esattamente "MPEG-1/2 Audio Layer 3") è un algoritmo di compressione audio in grado di ridurre drasticamente la quantità di dati richiesti per riprodurre un suono, rimanendo comunque una riproduzione fedele del file originale non compresso. È lo standard de facto della compressione audio.

La qualità di un file MP3 dipende dalla qualità della codifica e dalla difficoltà con il quale il segnale deve essere codificato. Buoni codificatori hanno una qualità accettabili da 128 a 160 kbit/s, la chiarezza perfetta di un brano si ottiene da 160 a 192 kbit/s. Un codificatore che ha bassa qualità lo si riconosce ascoltando persino un brano a 320 kbit/s. Per questo non ha senso parlare qualità di ascolto di un brano di 128 kbit/s o 192 kbit/s. Una buona codifica MP3 a 128 kbit/s prodotta da un buon codificatore produce un suono migliore di un file MP3 a 192 kbit/s codificato con uno scarso codificatore.

Oggi esistono buone alternative a MP3 anche se molto meno diffuse.

#### Vorbis

VORBIS è un algoritmo di compressione orientato alla compressione mono, stereo o 5.1 surround di segnale audio PCM campionato a 44.1 kHz o 48

---

kHz, con una profondità di campionamento di 16 bit o 32 bit. È comunque in grado di gestire anche segnali in ingresso differenti da quelli raccomandati.

Nel trattamento di segnale stereo musicale Vorbis ha il suo bit rate ideale intorno ai 128 kbit/s, risultando estremamente difficoltoso da distinguere rispetto all'originale in un ascolto cieco già da 192 kbit/s.

Trattandosi di un algoritmo di compressione lossy, cioè a perdita di informazioni, è l'encoder a svolgere il compito più delicato in assoluto, dovendo scegliere *quale* parte di informazione acustica sacrificare.

Vorbis è un algoritmo dall'approccio pesantemente VBR, ovvero a bit rate estremamente variabile in base al tipo di segnale sonoro che è chiamato a codificare. Per questo motivo al posto di riferirsi al valore di kbit/s Xiph.Org raccomanda di usare la nomenclatura *q*, ovvero il livello di qualità con cui è stato eseguita la codifica.

Tra i pregi che comunemente si attribuiscono a Vorbis, soprattutto riferendosi all'inevitabile paragone con lo standard de-facto MP3, vanno ricordati la maggior estensione e pulizia delle alte frequenze (sopra i 16 kHz), il supporto multicanale a livello nativo e in generale una migliore conservazione delle microinformazioni di spazialità sonora del segnale originario.

Tra i difetti attribuiti vanno citati la relativa pesantezza dell'algoritmo di decodifica rispetto al collaudato MP3, e soprattutto una certa tendenza al pre-echo, ovvero un'innaturale *fantasma sonoro* che sembra precedere di alcuni brevi istanti ogni brusco aumento di pressione sonora. L'esempio che tipicamente viene portato è quello di una sonata di pianoforte con attacchi di *fortissimo* dal silenzio, oppure il suono delle nacchere.

#### Advanced Audio Coding

Il formato Advanced Audio Coding (AAC) è un formato di compressione audio creato dal consorzio MPEG e incluso ufficialmente nell'MPEG-4. L'AAC fornisce una qualità audio superiore al formato MP3 con una codifica più compatta. Attualmente viene utilizzato principalmente da Apple nei suoi prodotti dedicati all'audio, difatti Apple usa una variante dell'AAC che gestisce i diritti d'autore per vendere musica attraverso il proprio negozio di musica on-line iTunes Music Store. Una compressione a 128Kbps, lo standard di iTunes Music Store corrisponde a circa 160 kbps di un mp3 a bitrate variabile.

Vari studi ed esperimenti, hanno mostrato come un AAC a 128Kbps, a differenza del formato mp3, sia pressoché identica a quella del cd originale.

Altri algoritmi di compressione lossy sono: Windows Media Audio (WMA), molto diffuso sui sistemi Windows; Dolby Digital (AC3) che può comprimere

---



fino a 6 canali audio, di cui 5 a piena larghezza di banda ed uno per gli effetti a bassa frequenza (LFE), fino a 384 kbit/s. Viene utilizzato nei DVD e nel sistema americano ATSC DTV, MPC Musepack è un formato opensource con una qualità maggiore dell'mp3 a parità di bitrate a scapito però delle dimensioni finali del file.

### Nota sul bitrate

I file multimediali sono per loro natura connessi al tempo che scorre. In altri termini ad ogni secondo è associato un certo contenuto informativo e quindi una certa sottosequenza di cifre binarie. Il numero di cifre binarie che compongono queste sottosequenze è detto bitrate. In altre parole il bitrate è il numero di cifre binarie impiegate per immagazzinare un secondo di informazione. Questo può essere costante per tutta la durata del file o variare all'interno di esso. Ad esempio i cd musicali vengono campionati (registrati) ad una frequenza pari a 44.100Hz. Da ciò si evince che ogni secondo si hanno 44.100 valori registrati dall'ipotetico microfono che vanno poi moltiplicati per i 2 canali del suono stereo che vanno a loro volta moltiplicati per 2 poiché la registrazione avviene a 16 bit (pari appunto a 2 byte). Quindi avremo:

$$44.100 \times 2 \times 2 \times 60 \text{ (secondi)} = \sim 10 \text{ MB ogni minuto}$$

La compressione, diminuendo la lunghezza globale del file, diminuirà di conseguenza la lunghezza media delle sottosequenze ossia diminuirà il bitrate medio. Il bitrate medio diventa dunque in questi casi l'indice dell'entità della compressione. Ad esempio se il file di origine possedesse un bitrate di 1411 Kbit/s e il file compresso possedesse un bitrate medio di 320 Kbit/s, allora avremmo ridotto di un fattore pari a circa 4.5.

## Compressione dei filmati

La codifica digitale e la compressione dei video è affidata ai cosiddetti codec video. Un CODEC VIDEO è un programma o un dispositivo sviluppato per descrivere un flusso video sotto forma di dati numerici adatti ad essere memorizzati su un supporto digitale. Usualmente i codec video effettuano anche una compressione dei dati in modo da ridurre l'elevata quantità di dati che compone un flusso video. La maggior parte dei codec video adottano tecniche di compressioni lossy (a perdita di informazioni) in modo da poter ridurre i dati necessari per trasmettere i flussi video anche di 20 volte o più, ma esistono anche dei codec utilizzati per applicazioni professionali che utilizzano compressioni lossless (senza perdita di informazione).

A seconda della diversa tecnica di codifica del flusso video, i codec video si dividono in due grandi famiglie: a codifica intraframe e a codifica interframe.

La codifica intraframe contraddistingue i codec che codificano e decodificano un flusso video descrivendo ogni singolo fotogramma che compone la sequenza video, rispettando quindi un approccio tradizionale alla quantizzazione video come sequenza di immagini statiche.

Nella codifica interframe invece i codec video si occupano di descrivere i cambiamenti che occorrono tra un fotogramma ed il successivo partendo da un fotogramma iniziale descritto con codifica intraframe e seguendo un approccio più innovativo alla quantizzazione video allo scopo di migliorarne l'efficienza sfruttando la capacità dei sistemi di riproduzione moderni in grado di elaborare l'informazione per poi mostrarne il risultato.

La diversa dinamica dei due approcci fa sì che la codifica intraframe è più adatta alla riproduzione di sequenze video particolarmente movimentate, descrivendo ogni singolo fotogramma infatti, un codec a codifica intraframe potrà degradare la qualità delle singole immagini all'aumentare del rapporto di compressione, ma tenderà comunque a lasciare inalterata la dinamica del movimento. I codec a codifica interframe risultano invece meno adatti alla codifica di sequenze movimentate per le quali necessitano di descrivere grossi cambiamenti tra i fotogrammi. Al contrario, in sequenze video statiche, ovvero con pochi elementi che cambiano nella scena, la codifica interframe risulta di notevole efficienza.

Ecco una breve panoramica dei principali standard:

- Il **DivX®** è un formato di compressione video sviluppato da DivX Inc. Attraverso l'apposito codec è possibile riprodurre e creare file video di questo formato. La particolarità del DivX sta nella sua versatilità nel produrre file di dimensioni ridotte di filmati di lunga durata, lasciando pressoché inalterata la qualità dell'immagine. In pratica, con le opportune impostazioni, è possibile convertire un film DVD di 6-8 Gigabyte in un file DivX di 700Mb (la dimensione di un cd rom) con una qualità video e audio più che discreta. Per questo motivo, è stato al centro di controversie per il suo utilizzo nella duplicazione e distribuzione di DVD protetti. Le prime versioni di DivX (fino alla 3) erano illegali in quanto n quanto prodotto da una copia rubata del Mpeg-4 di proprietà Microsoft. Dalla versione 4 DivX è diventato un codec indipendente dal suo predecessore ed iniziata la sua vita anche commerciale. Oggi DivX è arrivato alla versione 6.
  - H.264 Questo codec video è stato sviluppato per video ad alta qualità anche a frequenze di trasmissione dei dati inferiori rispetto alle soluzioni attuali, ed è utilizzata per qualunque tipo di periferica: dai televisori ad alta definizione HDTV e DVD, ai telefoni cellulari 3G. I servizi di broadcast basati sullo standard H.264 occupano una banda inferiore rispetto al diffuso schema di codifica MPEG-2, a una frequenza di trasmissione dei bit decisamente inferiore. Gli operatori di broadcasting possono quindi trasmettere in modo economico un
-

numero maggiore di programmi ad alta definizione. L'efficienza della compressione è migliorata di oltre il 50% rispetto al precedente MPEG-2. Attualmente i dispositivi con maggior diffusione ad utilizzare questo sistema di codifica sono l'iPod video e la console Sony PSP.

- MPEG-1 è uno standard introdotto nel 1991 da MPEG (Moving Pictures Experts Group). Originariamente è stato ottimizzato per le applicazioni video a basso *bitrate* con una risoluzione video di 352x240 pixel con 30 fotogrammi al secondo per lo standard tv NTSC oppure di 352x288 pixel con 25 fotogrammi al secondo per lo standard tv PAL. MPEG-1 non è strutturalmente limitato a questi formati in quanto può raggiungere ad esempio i 4095x4095 pixel con 60 fotogrammi al secondo, ma di fatto il sistema è stato ottimizzato per un bit rate di 1,5 Mbit/s. I Video CD utilizzano il formato MPEG-1. La qualità dell'output ai *bitrate* tipici di un Video CD è quella di un VCR.
  - MPEG-2 è un sistema di codifica digitale di immagini in movimento, che permette di comprimere i dati mantenendo una buona qualità. MPEG-2 è stato destinato al broadcast televisivo, fin dalla sua introduzione nel 1994. Una efficiente codifica per il video interlacciato e la scalabilità sono state le caratteristiche che hanno permesso di digitalizzare efficacemente i segnali televisivi. Grazie all'MPEG-2 si ottengono immagini televisive di buona qualità con bitrate compresi tra 4 e 9 Mbit/s. MPEG-2 sta anche alla base dello standard DVD.
  - MPEG-4, presentato nel 1998, è il nome dato a un'insieme di standard per la codifica dell'audio e del video digitale sviluppati dall'ISO/IEC Moving Picture Experts Group (MPEG). L'MPEG-4 è uno standard utilizzato principalmente per applicazioni come la videotelefonia e la televisione digitale, per la trasmissione di filmati via Web, e per la memorizzazione su supporti CD-ROM. MPEG-4 supporta tutte le caratteristiche degli standard MPEG-1 e MPEG-2 oltre a tutta una serie di nuove caratteristiche come la gestione tridimensionale degli oggetti (tramite un'estensione del VRML). I flussi audio e video vengono trattati dallo standard MPEG-4 come oggetti che possono essere manipolati e modificati in tempo reale. Lo standard supporta caratteristiche specificate da terze parti come una particolare gestione dei Digital Rights Management o una gestione interattiva dei contenuti.
  - REAL VIDEO è un video codec proprietario, sviluppato dalla RealNetworks. La sua prima release risale al 1997. REAL VIDEO è stato usato inizialmente per servire video streaming attraverso le reti internet ad un basso bit rate verso personal computer. Lo
-

sviluppo delle connessioni verso la banda larga ha permesso in tempi recenti di offrire filmati con una maggior qualità. Inoltre, lo streaming consente di scaricare e vedere filmati di qualsiasi tipo su apparecchi cellulari. Real Video differisce dalle codifiche normali in quanto è un formato proprietario ottimizzato esclusivamente per lo streaming attraverso il protocollo (proprietario) PNA oppure tramite il Real Time Streaming Protocol. Può essere usato per scaricare e vedere video o per uno streaming live.

- Windows Media Video (WMV) è il nome generico per una serie di tecnologie proprietarie sviluppate da Microsoft per lo streaming di file video. Fa parte della piattaforma Windows Media. A partire dalla versione 7 (WMV1), Microsoft ha usato una sua versione modificata dello standard MPEG-4.
  - **XviD** è un codec video open source aderente allo standard MPEG-4 (profilo ASP) originariamente basato su OpenDivX. Il progetto XviD è partito nel Luglio 2001, con la chiusura del progetto OpenDivX da parte di DivXNetworks Inc., società creatrice del popolare codec DivX. Il codec XviD ha avuto una considerevole diffusione soprattutto nell'ambito del file sharing, dove viene utilizzato principalmente per comprimere la parte video dei film in modo che possano occupare poco spazio ed essere trasferiti velocemente nelle reti, appunto, di file sharing. La diffusione sul p2p di questo codec e il fatto che aderisse allo standard MPEG-4 ASP hanno fatto sì che molti produttori di lettori DVD che già erano in grado di leggere flussi video compressi in DivX (questi particolari lettori DVD spesso vengono chiamati *lettori Stand Alone*) aggiungessero il supporto ad XviD. La diffusione di XviD, oggi, è paragonabile a quella del codec "rivale" DivX.
-

## La programmazione

Abbiamo visto come le informazioni vengono codificate per potere essere elaborate dal computer. Ora passeremo a vedere gli strumenti che abbiamo a disposizione per programmare il computer, per fare in modo, cioè, che il computer elabori le informazioni che gli forniamo in modo utile per noi.

Prima di affrontare la programmazione in senso stretto faremo un breve panorama dello sviluppo che ha avuto negli anni la programmazione dei computer e degli strumenti che ci offre oggi.

### Un po' di storia

All'inizio, negli anni '40, l'unico metodo per programmare era il **linguaggio macchina**. Il lavoro del programmatore consisteva nel settare ogni singolo bit a 1 o 0 su enormi computer che occupavano stanze intere e pesavano decine di tonnellate. I monitor non esistevano; i dati e i programmi si fornivano al computer su schede perforate e il computer mandava i risultati su telescriventi. Questo tipo di procedimento, non solo era faticoso, ma era riservato ad una cerchia ristretta di persone. Eppure in questo modo gli scienziati riuscirono in pochi mesi a completare i calcoli per costruire la prima bomba atomica, calcoli che se fatti a mano, con calcolatrici meccaniche avrebbero richiesto anni.

Negli anni 50, con il progresso tecnologico e la riduzione delle dimensioni e dei costi dei calcolatori, nacquero due importanti linguaggi di programmazione, il **FORtrAN** (FORmula trANslator), il cui utilizzo era ed è prettamente quello di svolgere in maniera automatica calcoli matematici e scientifici, e l'**ALGOL** (ALGOrithmic Language), altro linguaggio per applicazioni scientifiche sviluppato da Backus (l'inventore del FORtrAN) e da Naur, i quali oltretutto misero a punto un metodo per rappresentare le regole dei vari linguaggi di programmazione che stavano nascendo.

Nel 1960 venne presentato il **COBOL** (COmmon Business Oriented Language), ideato per applicazioni nei campi dell'amministrazione e del commercio, per l'organizzazione dei dati e la manipolazione dei file.

Nel 1964 fa la sua comparsa il **BASIC**, il linguaggio di programmazione per i principianti, che ha come caratteristica fondamentale quella di essere

---

molto semplice e, infatti, diventa in pochi anni uno dei linguaggi più utilizzati al mondo.

Intorno al 1970, però, Niklaus Wirth pensò bene di creare il **PASCAL** per andare incontro alle esigenze di apprendimento dei neo-programmatori, introducendo però la possibilità di creare programmi più leggeri e comprensibili di quelli sviluppati in basic. Si può affermare con certezza che Wirth ha centrato il suo obiettivo, considerando che ancora oggi il Pascal viene usato come linguaggio di apprendimento nelle scuole.

Pochi anni più tardi fa la sua comparsa il **C** (chiamato così perché il suo predecessore si chiamava B), che si distingueva dai suoi predecessori per il fatto di essere molto versatile nella rappresentazione dei dati. Il C, infatti, ha delle solide basi per quanto riguarda la strutturazione dei dati, però può apparire come un linguaggio assai povero vista la limitatezza degli strumenti a disposizione. Invece la sua forza risiede proprio in questi pochi strumenti che permettono di fare qualsiasi cosa, non a caso viene considerato "il linguaggio di più basso livello tra i linguaggi ad alto livello", per la sua potenza del tutto paragonabile al linguaggio macchina, mantenendo però sempre una buona facilità d'uso.

Ma la vera rivoluzione si è avuta nel 1983 quando Bjarne Stroustrup inventò il **C++** (o come era stato chiamato inizialmente "C con classi") che introduceva, sfruttando come base il C, la programmazione Orientata agli Oggetti (OO - Object Oriented) usando una nuova struttura, la classe. La programmazione orientata agli oggetti consente di dividere l'interfaccia dal contenuto, ottenendo in questo modo tanti "moduli" interagibili tra loro attraverso le interfacce, permettendo così al programmatore di cambiare il contenuto di una classe (se sono stati trovati errori o solo per introdurre delle ottimizzazioni) senza per questo doversi preoccupare di controllare eventuale altro codice che richiami la classe. Questo nuovo stile di programmazione ha completamente stravolto il modo di programmare precedente ad esso che si riduceva ad una programmazione procedurale che lasciava poco spazio al riutilizzo del codice.

Insomma il C++ è riuscito a creare un nuovo modo di programmare, o meglio di progettare un programma, rendendo il codice scritto più chiaro e soprattutto "riutilizzabile", ed è grazie a lui che oggi possiamo usare le finestre colorate di Windows che ci piacciono tanto.

Il modello Object Oriented ha avuto grande successo e dopo il C anche il Pascal, il Basic e altri linguaggi hanno avuto la loro versione OOP. Negli ultimi anni, infine, è nato JAVA che ha due caratteristiche principali:

- È il primo linguaggio progettato appositamente per la programmazione orientata agli oggetti (non è cioè l'adattamento di un linguaggio preesistente).
-

- Il programma ottenuto non è un programma destinato a *girare* in un ambiente specifico (Windows o Macintosh o Linux) ma gira su un computer virtuale, la Java Virtual Machine. Basterà installare la versione specifica della Java Virtual Machine per il sistema operativo specifico e lo stesso programma potrà girare in ambienti completamente diversi.

Altri linguaggi di programmazione da menzionare sono il LISP (1959), l'ADA (1970), lo SMALLTALK (1970) e il LOGO e il PROLOG, ecc .

## Il panorama attuale

L'enorme diffusione dei personal computer ha influito anche sul mercato dei linguaggi di programmazione, una volta sicuramente un mercato marginale. Pensiamo solamente alla diffusione di HTML e dei linguaggi di scripting ad esso collegati e ci renderemo conto di quanta più gente rispetto a solo dieci anni fa oggi si avvicina in qualche modo al mondo della programmazione.

Da un lato i produttori di software hanno tentato di assecondare questo processo cercando di offrire prodotti sempre più semplici da usare. Sono così nate le versioni visuali di vari linguaggi di programmazione che rendono enormemente più semplice sviluppare programmi nei moderni ambienti grafici a finestre: Visual Basic, Visual C ++, Delphi (versione *visual* di Object Pascal), Kilyx (versione di Delphi per il mondo Unix), Visual Java, ecc.

Dall'altro gli strumenti per lo sviluppo di contenuti multimediali on-line e off-line (pagine web, animazioni interattive, ecc.) si sono dotati di linguaggi di programmazione sempre più potenti.

Il nostro interesse è soprattutto rivolto allo sviluppo di contenuti multimediali. Può essere, comunque interessante dare un rapido sguardo agli strumenti che abbiamo a disposizione anche per realizzare programmi orientati ad altri fini. Bisogna infatti tenere conto che i linguaggi di programmazione, per loro natura, possono fare benissimo alcune cose e male altre.

Per **Applicazioni Matematiche o di Ricerca** i linguaggi più adatti sono ancora i *vecchi* Fortran e Algol.

Per i grandi progetti software (sviluppo di sistemi operativi, applicativi complessi, applicazioni lato server) normalmente viene impiegato il C o il C++.

Per lo sviluppo di **programmi gestionali** in ambiente Windows lo strumento più utilizzato è Visual Basic prodotto da Microsoft. Un'ottima alternativa al Visual Basic è, da qualche anno Delphi un prodotto Borland basato sul Pascal che consente anche il porting dei programmi tra

---

l'ambiente Windows e l'ambiente Unix/Linux grazie a Kilyx versione Unix di Delphi. Negli ambienti diversi da Windows si utilizzano C/C++ e ambienti di sviluppo (i cosiddetti CASE) basati su C/C++.

## Applicazioni multimediali

Dal punto vista del nostro corso meritano una particolare attenzione gli strumenti per lo sviluppo di applicazioni multimediali on line e off line.

### Sviluppo di pagine web

L'attività di programmazione entra a vari livelli nello sviluppo di contenuti da distribuire via Internet.

Un primo livello di base è la costruzione delle pagine web. Il linguaggio usato è l'HTML (Hyper Text Markup Language) e descrive come una pagina web debba essere mostrata da un browser. L'HTML è un linguaggio a *marcatori*: tutte le istruzioni proprie del linguaggio sono inserite tra due segni specifici (nel caso di HTML tra "<" e ">") e costituiscono i cosiddetti tag. I browser cercano di interpretare i tag come istruzioni mentre il resto viene mostrato come testo. I tag che il browser non riesce ad interpretare vengono semplicemente ignorati.

Questo approccio estremamente flessibile ha consentita un grande sviluppo delle possibilità di HTML nelle varie versioni successive senza la necessità di preoccuparsi troppo della compatibilità con le versioni precedenti.

Specifici tag consentono poi di inserire nella pagine web vari tipi di componenti che a loro volta sono programmabili. Vediamo le più importanti:

- Il tag SCRIPT consente di inserire dei nuclei di codice che il browser è in grado di eseguire e che, ad esempio, possono servire ad aumentare il grado di interattività della pagina. I linguaggi a disposizione dello sviluppatore sono due: VBSCRIPT (che deriva dal Visual Basic e viene riconosciuto però solo da Internet Explorer) e JAVASCRIPT che deriva invece da JAVA ed è riconosciuto più o meno da tutti i browser.
  - Il tag APPLET che consente di inserire in una pagina web un programma scritto in JAVA. La condizione per cui il programma possa essere eseguito è che la Java Virtual Machine sia installata sul computer.
  - Il tag OBJECT (Internet Explorer su piattaforma Windows) che insieme al tag EMBED (altri browser e Internet Explorer su Macintosh) consentono di inserire nelle pagine web oggetti di varia natura gestiti da estensioni dei browser o (in Windows) dai oggetti gestiti direttamente dal sistema operativo (i cosiddetti ActiveX). Qui segnaliamo due oggetti/plugin che negli ultimi anno hanno avuto una grande diffusione: SHOCKWAVE FLASH e SHOCKWAVE
-



DIRECTOR entrambi orientati allo sviluppo di applicazioni multimediali interattive, entrambi dotati di un potente linguaggio di programmazione.

Un altro importante livello in cui l'attività di programmazione entra nello sviluppo di pagine web è la programmazione *lato server*.

I tipi di pagine web di cui abbiamo fin qui parlato sono pagine statiche. Pagine, cioè, che vengono inviate al browser dal server web esattamente come sono state composte. Molti siti web oggi usano, invece, pagine dinamiche, pagine, cioè, il cui contenuto viene composto dal server al momento della richiesta sulla base dei parametri che gli vengono passati dal browser e del contenuto di database che il server può consultare.

Questo tipo di programmazione, un tempo riservato agli specialisti, è oggi alla portata di qualsiasi programmatore. Gli strumenti più diffusi sono PERL e PHP per i server web che girano in ambiente Unix e Linux e ASP e la sua più recente evoluzione ASP.NET per Internet Information Server (il server web Microsoft).

Applicazioni multimediali off line

Per quanto riguarda la Creazione di Giochi gli sviluppatori tendono a usare (per ottenere la massima efficienza) strumenti il più vicino possibile al linguaggio dei processori quindi C++ e Assembler. Per giochi non troppo complessi, però, anche FLASH, DIRECTOR e JAVA possono essere buone soluzioni.

FLASH è probabilmente il prodotto più abbordabile per i neofiti. offre una potente gestione della grafica e delle animazioni ed è in grado di produrre applicazioni molto leggere che possono essere facilmente distribuiti anche su Internet. Le ultime versioni (dalla 7.0 in poi) di Flash sono dotate di linguaggio molto potente ed evoluto (ACTION SCRIPT 2.0 derivato da Javascript) che consente di utilizzare tecniche di Object Oriented Programming.

DIRECTOR è l'ideale per produrre applicazioni multimediali off line multiplatforma (Windows e Macintosh). Offre un linguaggio di programmazione molto potente ed è in grado di gestire molto bene risorse multimediali diverse (suono, video, immagini vettoriali e bitmap, filmati flash, ecc.). Nelle ultime versioni, inoltre, lo sviluppatore può scegliere se utilizzare la *vecchia* sintassi del LINGO (linguaggio di programmazione proprietario sviluppato appositamente per Director) o la sintassi di JAVASCRIPT (sintassi derivata da Java, la stessa utilizzata nella programmazione dei browser e da Action Script).

Concludiamo questa breve panoramica spendendo qualche parola su JAVA. Il punto di forza di JAVA è la portabilità. Se sviluppate correttamente le applicazioni Java possono girare su un'infinità di dispositivi per cui (in forme

---

diverse) è stata sviluppata la Java Virtual Machine (Windows, Macintosh, Unix, Linux, telefonini, palmari, ecc.).

# Algoritmi

## Che cos'è un programma?

Un computer è una macchina in grado di eseguire un determinato set di istruzioni. Per poter svolgere un compito il calcolatore ha bisogno di qualcosa che gli dica passo passo cosa deve fare, o meglio, quale delle istruzioni che conosce deve eseguire in un determinato momento.

Un programma è un compito che il computer deve svolgere tradotto in una serie di istruzioni che il calcolatore è in grado di eseguire. L'elaboratore è in grado di eseguire istruzioni molto basiche o, come si dice, a basso livello. Un programma scritto direttamente nel linguaggio che il computer è in grado di capire sarebbe per noi quasi incomprensibile. Ci aiutano a scrivere programmi per noi comprensibili i linguaggi di programmazione.

Dato un problema o un compito è necessario quindi "tradurlo" in termini di istruzioni nel linguaggio che abbiamo scelto. Ma prima di iniziare a programmare è conveniente analizzare il nostro problema e individuarne la soluzione in maniera indipendente dal linguaggio che useremo per programmare. Descrivere la soluzione di un problema o lo svolgimento di un compito sotto forma di passi discreti, eseguibili e non ambigui significa costruire un ALGORITMO.

## L'algoritmo

Un ALGORITMO si può definire come un *procedimento* che consente di *ottenere* un dato *risultato* eseguendo, in un determinato ordine, un insieme di *passi semplici* corrispondenti ad azioni scelte solitamente da un insieme finito.

Le proprietà fondamentali di un algoritmo sono: non ambiguità (il procedimento deve essere interpretabile in modo univoco da chi lo deve eseguire), eseguibilità (ogni istruzione dell'algoritmo deve poter essere eseguita senza ambiguità da parte di un esecutore reale o ideale), finitezza (sia il numero di istruzioni che compongono l'algoritmo, che il tempo di esecuzione devono essere finiti).

---

Prendiamo il noto problema del contadino che deve far attraversare il fiume ad una capra, un cavolo e un lupo, avendo a disposizione una barca in cui può trasportare solo uno dei tre alla volta. Se incustoditi, la capra mangerebbe il cavolo e il lupo mangerebbe la capra.

Ecco l'algoritmo che descrive la soluzione:

```
1. Inizio
2. Porta la capra sull'altra sponda
3. Porta il cavolo sull'altra sponda
4. Riporta la capra sulla sponda di partenza
5. Porta il lupo sull'altra sponda
6. Porta la capra sull'altra sponda
7. Fine
```

Come secondo esempio vediamo l'algoritmo che sta alla base di qualsiasi computer e che stabilisce come funziona la CPU:

```
finché non trovi un'istruzione di halt fai:
  preleva un'istruzione dalla memoria
  decodifica l'istruzione
  esegui l'istruzione
```

Questo algoritmo è intrinseco nella struttura della CPU e fa in modo che una volta inserito un programma nella memoria, questo possa essere letto istruzione per istruzione ed eseguito. Tale caratteristica sta alla base della programmabilità e permette la totale generalità degli elaboratori.

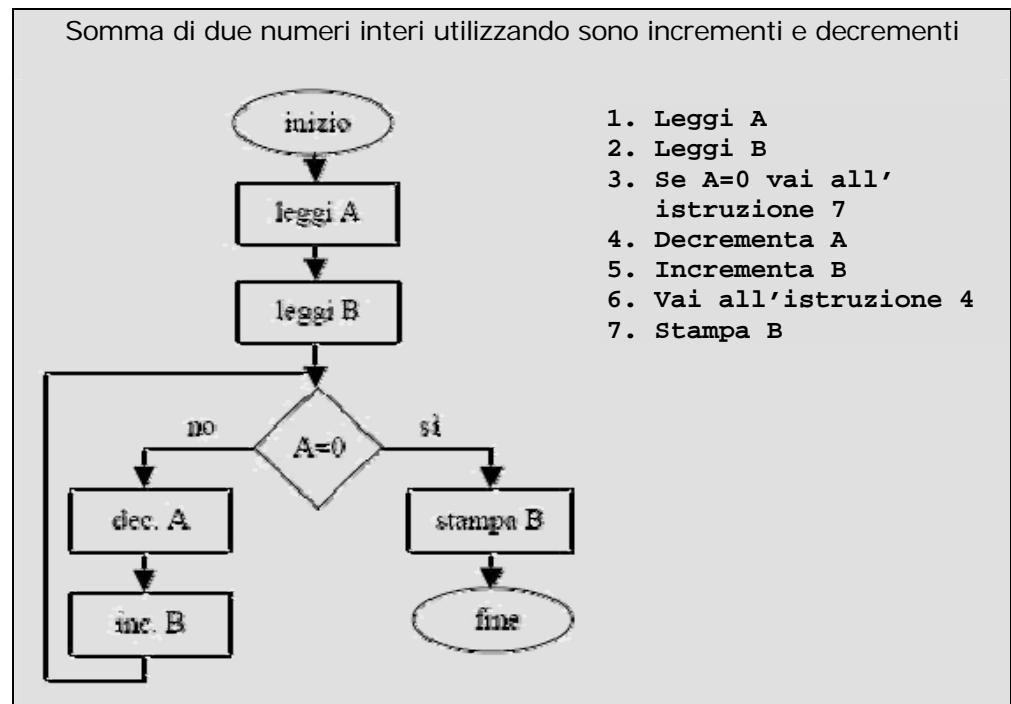
## Rappresentazione di un algoritmo

Al fine di rappresentare gli algoritmi in modo comprensibile per il programmatore, vengono principalmente utilizzate due tecniche:

- Rappresentazione mediante pseudolinguaggio o pseudocodice che possiamo definire come un *linguaggio informale (ma non ambiguo)* per la descrizione delle istruzioni. Ogni riga rappresenta un'istruzione. Se non diversamente specificato, le righe si leggono dall'alto verso il basso.
- rappresentazione mediante diagramma di flusso (flowchart) che è una rappresentazione grafica in cui ogni istruzione è descritta all'interno di un riquadro e l'ordine di esecuzione delle istruzioni è indicato da frecce di flusso tra i riquadri.

L'esempio che segue mostra le due rappresentazioni dell'algoritmo che calcola la somma di due numeri interi usando esclusivamente istruzioni di incremento e decremento.

---



## Definizione di uno pseudolinguaggio

Uno pseudolinguaggio si basa sul concetto di azione primitiva. L' utilizzo di azioni primitive per descrivere un procedimento fa sì che questo possa essere compreso senza alcuna ambiguità. In altre parole, nel caso in cui si descriva una azione in modo superficiale o si utilizzino vocaboli dal significato non univoco una frase potrebbe essere interpretata in diversi modi. Si prenda, per esempio, la frase *"andare a lezione può essere irritante"*. Un lettore che non conosca gli argomenti della lezione, il docente o il luogo in cui la lezione si svolge potrebbe interpretare la frase come: *"La lezione causa irritazione"* oppure *"il tragitto per arrivare dove si svolge la lezione è irritante"* oppure in entrambi i modi.

Per evitare un' interpretazione ambigua, quindi, viene definito un set di attività primitive, note agli interlocutori (uomo – macchina) e un insieme di regole per utilizzare le primitive. Tutto questo, set di primitive più l'insieme delle regole, costituisce un linguaggio di programmazione.

Ogni pseudolinguaggio deve definire un numero minimo di elementi quali:

1. Istruzioni di **start, end**  
inizio e di fine

Sono i punti d' ingresso e di uscita del flusso di esecuzione

2. Istruzione di **nome ← espressione**  
assegnamento

L' esecuzione di un' istruzione di assegnamento avviene in due fasi: la valutazione dell' espressione (utilizzando i valori correnti delle eventuali variabili che in essa compaiono) e l' aggiornamento del valore della variabile a sinistra del segno

3. Scelta tra due possibili attività	<b>if</b> (condizione) <b>then</b> (attività 1) <b>else</b> (attività 2)	Biforcazione del flusso in due percorsi alternativi (mutuamente esclusivi) la cui esecuzione è condizionata al verificarsi di una condizione. L' esecuzione di un' istruzione condizionale comporta innanzitutto la valutazione della condizione, e quindi l' esecuzione delle istruzioni che compongono uno dei due cammini.
4. Strutture iterative	<b>while</b> (condizione) <b>do</b> (azione)	Esecuzione ripetuta di una o più istruzioni. La ripetizione dipende dall' esito di un controllo. È fondamentale che l' esito del controllo dipenda da variabili il cui valore può essere modificato dall'esecuzione delle istruzioni del ciclo. In caso contrario il ciclo verrebbe eseguito o mai (risultando inutile) o all' infinito (violando la definizione di algoritmo). Le variabili da cui dipende il controllo sono dette variabili di controllo del ciclo. Ogni ciclo è composto da 4 elementi: l' inizializzazione delle variabili di controllo, il controllo della condizione da cui dipende l' iterazione, il corpo del ciclo (sequenza delle istruzioni da eseguire ciclicamente) e l' aggiornamento delle variabili di controllo.
5. Istruzione di salto	<b>goto</b> (riga di destinazione)	L'istruzione di salto sposta il flusso di esecuzione in un particolare punto dell'algoritmo. La riga di destinazione rappresenta, appunto, il punto in cui l' esecuzione riprenderà dopo il goto.
6. Definizione di procedure	<b>procedure</b> nome ( istruzione 1 istruzione 2 istruzione 3 )	Unità di programma utilizzabili attraverso invocazione da un qualsiasi punto del codice. Tutti i linguaggi di programmazione utilizzano questa strategia per rendere più leggibile il codice. Sinonimi di procedura sono: funzione, metodo, subroutine, sottoprogramma ecc

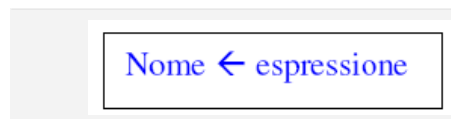
## Diagramma di flusso

Il flusso inizia e termina in blocchi fittizi (detti inizio e fine) di forma generalmente arrotondata. I riquadri del diagramma di flusso possono avere forme diverse (convenzionali) per rappresentare graficamente il tipo di istruzione che contengono. Normalmente si usano solo tre tipi di riquadri: rombi con una freccia entrante e due uscenti per indicare diramazioni di flusso condizionate, rettangoli per indicare qualsiasi istruzione che non sia una condizione, ellissi per indicare inizio e fine.

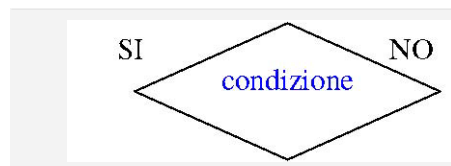
1. Inizio e fine



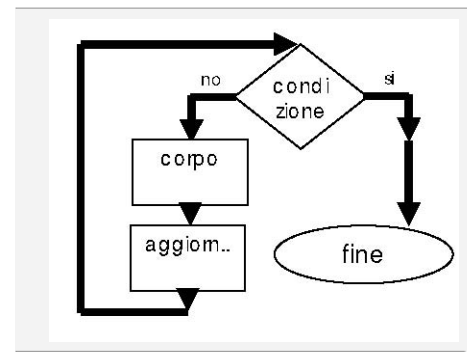
2. Assegnamento ed elaborazione



3. Scelta condizionata



## 4. Iterazione e salti



## Esecuzione dell'algoritmo

Un algoritmo può essere visto come un risolutore di problemi che acquisisce dati d'ingresso e produce risultati. Generalmente, uno stesso algoritmo può essere applicato più volte per risolvere lo stesso tipo di problema con dati d'ingresso diversi, ovvero istanze diverse dello stesso problema. Chiamiamo esecuzione di un algoritmo la sua applicazione a determinati dati d'ingresso. Un algoritmo è detto lineare se l'esecuzione segue un flusso lineare dalla prima all'ultima istruzione, altrimenti è detto non lineare. (Es: L'algoritmo per la somma di due numeri naturali dell'esempio visto in precedenza è non lineare). Chiamiamo passo d'esecuzione l'esecuzione di un'istruzione. Il numero di passi d'esecuzione è generalmente diverso dal numero di istruzioni utilizzate per descrivere l'algoritmo e dipende dai dati d'ingresso.

Durante l'esecuzione dell'algoritmo entrano in gioco tre tipi di grandezze:

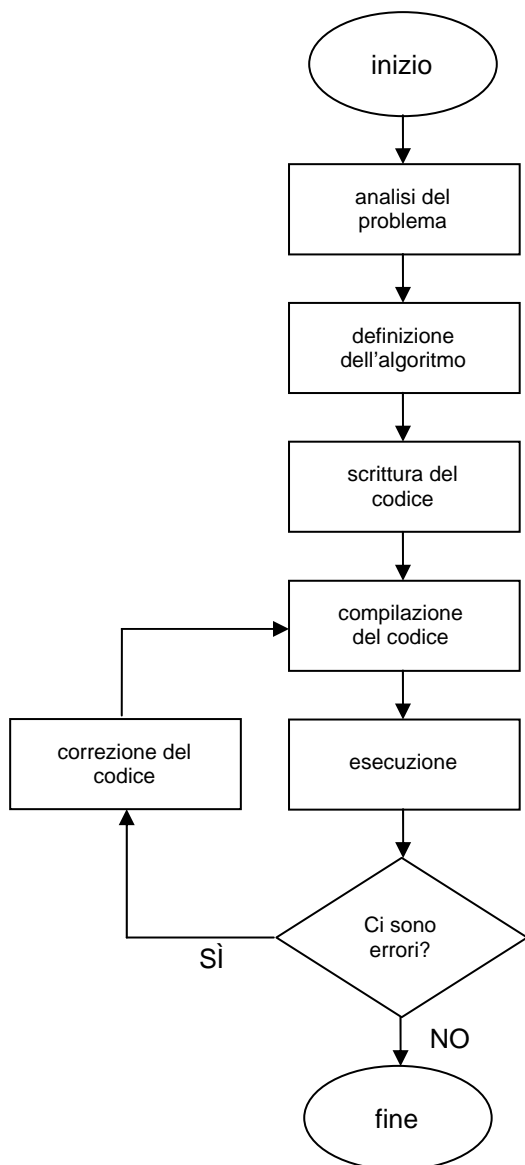
- Costanti: le costanti sono quantità note a priori il cui valore non dipende dai dati d'ingresso e non cambia durante l'esecuzione dell'algoritmo (ad esempio lo 0 dell'istruzione 4 dell'algoritmo per la somma di due numeri interi descritto precedentemente).
- Variabili: le variabili sono nomi simbolici cui sono assegnati dei valori che possono dipendere dai dati d'ingresso e cambiare durante l'esecuzione dell'algoritmo ad esempio A e B nell'algoritmo per la somma di due numeri naturali descritto precedentemente).
- Espressioni: un'espressione è sequenza di variabili, costanti o espressioni combinate tra loro mediante operatori (ad esempio  $a+b*4$ ). Nella valutazione di un'espressione si sostituisce ad ogni variabile il suo valore corrente e si seguono le operazioni secondo regole di precedenza prestabilite (o indicate da parentesi).

Una volta costruito l'algoritmo, definita cioè in maniera chiara la strategia con cui il nostro programma affronterà il problema che deve risolvere possiamo passare alla scrittura vera e propria.

## La realizzazione del programma

Qualsiasi sia il linguaggio che abbiamo scelto (anche l'ASSEMBLER) il nostro codice dovrà essere tradotto, perché possa essere eseguito dal computer, in una serie di istruzioni che la CPU è in grado di interpretare ed eseguire. Questo processo, che chiameremo compilazione, a secondo dell'ambiente e del linguaggio scelto potrà avere diverse caratteristiche ma sarà comunque un passaggio necessario del nostro processo creativo.

Se utilizziamo un diagramma di flusso per descrivere il processo di creazione di un'applicazione otteniamo qualcosa del genere:





In sintesi la creazione di un'applicazione, anche semplice, passa attraverso tre fasi: la progettazione (analisi del compito che vogliamo far svolgere all'applicazione e definizione dell'algoritmo che fa in modo che un elaboratore possa svolgere quel compito, la scrittura del codice nel linguaggio di programmazione che abbiamo scelto e il debugging (verifica del buon funzionamento dell'applicazione, correzione degli eventuali errori o variazioni che ne migliorino il funzionamento).



## Impariamo ad Indentare

Scrivere programmi *sensati* e *leggibili* è difficile, ma molto importante. Un primo passo (quello che abbiamo tentato di descrivere nel capitolo precedente) è dedicare il tempo necessario alla progettazione della nostra applicazione. Questa fase ci aiuterà a chiarire la logica e la sintassi del nostro lavoro.

Il secondo passo è scrivere il programma in maniera leggibile, ma non dal punto di vista sintattico, bensì facendo attenzione a come quello che scriviamo viene presentato visivamente. Se la forma che diamo al nostro codice è piacevole e, soprattutto, chiara, sarà più facile ricordarsi che compito quel pezzo di codice aveva nell'economia del nostro programma e sarà più facile individuare errori sia sintattici che semantici. Non esiste compilatore che possa dirci gli errori "logici" che stiamo facendo.

La chiarezza della scrittura si ottiene attraverso due *tecniche* molto semplici ma estremamente utili.

La prima tecnica è l'indentazione e consiste solamente nell'inserire spazi o tabulazioni (generalmente ignorati dal compilatore) per mettere subito in luce eventuali gerarchie dei cicli o delle funzioni.

Ad esempio creando una semplice tabella in HTML possiamo scriverla in questo modo:

```
<table> <tr> <td>a</td> <td>b</td> <td>c</td> </tr> <tr> <td>  
<table> <tr> <td>a1</td> </tr> <tr> <td>a2</td> </tr>  
</table> </td> <td>b1</td> <td>c1</td> </tr> </table>
```

ma, francamente, non si capisce esattamente come verrà presentato il testo nel documento HTML; per questo con l'inserimento di alcuni spazi si riesce a rendere più chiaro per noi e per gli altri quello che stiamo scrivendo,

```
<table>  
  <tr>  
    <td>a</td>  
    <td>b</td>  
    <td>c</td>  
  </tr>  
<tr>
```

```
<td>
<table>
  <tr>
    <td>a1</td>
  </tr>
  <tr>
    <td>a2</td>
  </tr>
</table>
</td>
<td>b1</td>
<td>c1</td>
</tr>
</table>
```

La seconda tecnica è quella di anche di "commentare" il codice scritto in maniera da rendere più chiare le operazioni logiche che stiamo svolgendo. Possiamo commentare un qualsiasi codice in svariati modi, qui sotto ne elenchiamo alcuni di esempio presi dai più comuni linguaggi di programmazione:

```
//  o  #
```

Posto dopo un'istruzione, tutto ciò che compare sulla stessa linea dopo questi simboli verrà interpretato come commento.

```
/*  */  o  <!--  -->
```

Ogni carattere compreso tra questi simboli verrà interpretato come commento

Inoltre poiché qualsiasi monitor è limitato in larghezza sarebbe buona norma, quando si scrive un programma od anche un semplice file HTML, non superare, mentre si scrive, le 80/90 colonne, poiché oltre tale limite siamo costretti ad usare la scrollbar dell'editor per continuare a leggere il testo, perdendo così del tempo e rischiando di perdere il filo logico del documento.

---

## Introduzione alla logica

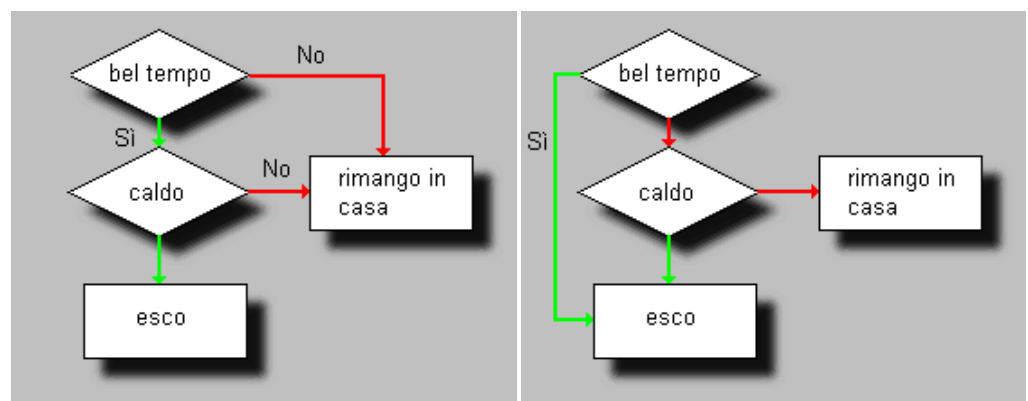
### L'algebra di Boole

George Boole, nel 1854, pubblicò un libro, *An Investigations of the Laws of Thought* (Un esame sulle leggi logiche del pensiero), in cui dimostrava che la maggior parte del pensiero logico, privata di particolari irrilevanti e verbosità, potesse essere concepita come una serie di scelte. Questa idea è divenuta la base dei computer.

In questa presentazione l'algebra di Boole, piuttosto che con il suo formalismo matematico, verrà proposta in modo da renderla più simile al linguaggio naturale: in tal modo, risulterà più intuitivo comprendere il funzionamento di quei semplici circuiti digitali che costituiscono la base dei computer.

Come abbiamo visto nei capitoli precedenti qualsiasi processo logico può essere ricondotto ad una sequenza di eventi elementari, che nell'insieme prende il nome di algoritmo e tale sequenza può essere rappresentata con un diagramma di flusso (il quale a sua volta è facilmente traducibile in un particolare programma comprensibile dall'elaboratore).

Prenderemo le mosse per la nostra discussione riferendoci ad un "problema" del tutto comune. Prendiamo questi due enunciati: "esco se è bel tempo ed è caldo"; "esco se è bel tempo o se è caldo".



E' facile riconoscere che le due decisioni sono distinte: la prima (diagramma a sinistra), comporta il verificarsi di due condizioni (evidenziate in verde); la

seconda (diagramma a destra), comporta il verificarsi di almeno una fra due condizioni. E' importante rendersi conto che, come vedremo, l'elaboratore non "comprende" il significato delle frasi "esco... resto in casa": si limita a considerare il valore di certe costanti associate ad ogni singolo evento.

Come secondo passo, si tratta di convertire i diagramma di flusso in un linguaggio numerico, il solo comprensibile dall'elaboratore. Ciò si ottiene con i cosiddetti operatori logici elementari. Per semplicità, limiteremo la nostra discussione ai tre elementi di base.

Con le istruzioni riportate nella tabella a fianco, possiamo tradurre i due differenti diagrammi di flusso in sequenze di istruzioni.

operazione	istruzione	porta logica
controllo	IF (se...)	
azione	THEN (allora...)	
coniunzione	AND (...e...)	AND
separazione	OR (... oppure...)	OR
negazione	NOT (negazione)	NOT

Per far questo, è necessario aggiungere un nuovo simbolo grafico (un parallelogramma) di inizializzazione ai nostri diagrammi di flusso. Questo simbolo implica che l'elaboratore si attende che gli vengano forniti i valori di  $A$  e  $B$  (che possono essere 0 o 1) tramite tastiera. Appena inseriti questi valori (per es.  $A=1$  e  $B=1$ ), l'elaboratore esegue il programma tenendo conto dei valori di inizializzazione ( $C=1$  "esco";  $C=0$  "rimango in casa"): a seconda del risultato ottenuto, sullo schermo verrà mostrata una delle due frasi.

$A = 1$  corrisponde all'evento "bel tempo"

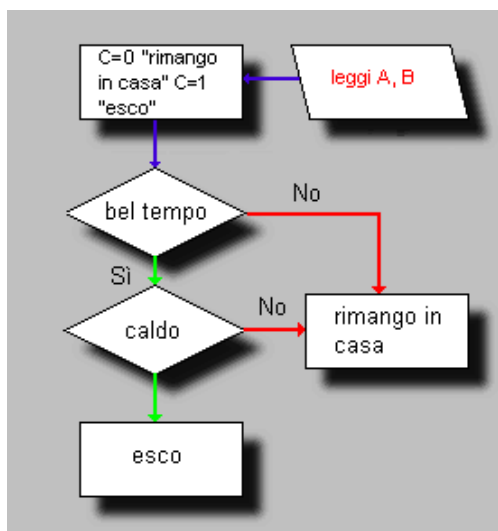
$B = 1$  corrisponde all'evento "caldo"

$C = 1$  corrisponde all'azione "esco"

$A = 0$  corrisponde all'evento "non bel tempo"

$B = 0$  corrisponde all'evento "non caldo"

$C = 0$  corrisponde all'azione "resto in casa"



con queste condizioni, il primo diagramma di flusso risulta così formalizzato:

**IF  $A$  AND  $B$  THEN  $C$**

Le istruzioni **AND** e **OR**, dette operatori logici, sono prese dall'algebra di Boole e forniscono il risultato 1 o 0, a seconda del valore delle variabili  $A$  e  $B$ . Ad esempio, se entrambe le variabili  $A$  e  $B$  valgono 1, allora l'operatore AND assumerà il valore 1. In questo caso, il simbolismo, tradotto in parole, corrisponde a:

**SE** *bel tempo* **E** *caldo* **ALLORA** *esco*

viceversa, se una sola delle due variabili, oppure entrambe valgono 0, allora l'operatore AND assumerà il valore 0. In questo caso, il simbolismo, tradotto in parole, corrisponde a:

**SE** *non bel tempo* **E** *caldo* **ALLORA** *resto in casa*  
**SE** *bel tempo* **E** *non caldo* **ALLORA** *resto in casa*  
**SE** *non bel tempo* **E** *non caldo* **ALLORA** *resto in casa*

Come abbiamo detto, i valori delle variabili  $A$  e  $B$  sono introdotti da tastiera prima di "raggiungere" l'operatore AND. Così, se entrambe le variabili  $A$  e  $B$  valgono 1, quando saranno esaminate dall'operatore AND seguirà necessariamente il risultato  $C = 1$ .

Con le stesse modalità, il secondo diagramma di flusso risulta così formalizzato:

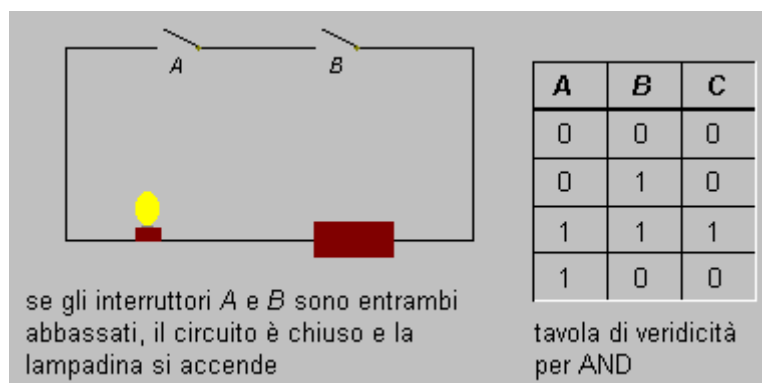
**IF  $A$  OR  $B$  THEN  $C$**

In questo caso, se almeno una delle variabili  $A$  e  $B$  vale 1, allora l'operatore OR assumerà il valore 1; viceversa, se entrambe valgono 0, allora l'operatore OR assumerà il valore 0. Dunque, se almeno una delle variabili  $A$  e  $B$  vale 1, quando saranno esaminate dall'operatore OR seguirà necessariamente il risultato  $C = 1$ .

Ora che abbiamo visto come definire un linguaggio numerico (ricordiamo che l'elaboratore *non* comprende il significato delle frasi che leggiamo) interpretabile dall'elaboratore, vediamo come applicarlo. Per far questo, esaminiamo un semplice "elaboratore" in grado di automatizzare la nostra decisione. La realizzazione di questo - e di tutti gli elaboratori - richiede la costruzione di circuiti detti porte logiche.

---

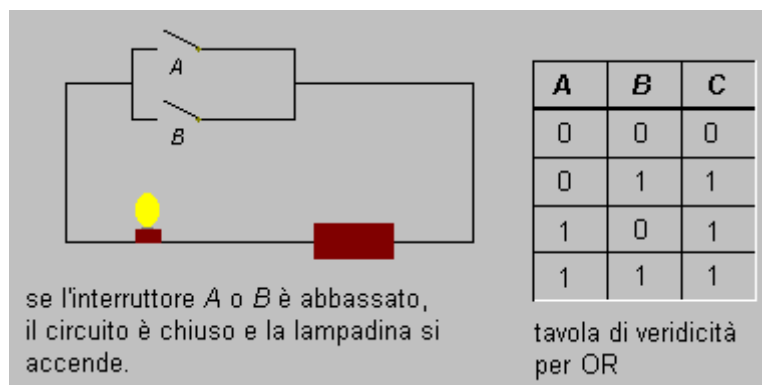
Per esempio, osservando il circuito elettrico schematizzato in figura, è facile riconoscere che la lampadina si accenderà solo se entrambi gli interruttori A



e B verranno abbassati in modo da chiudere il contatto elettrico. Questo circuito, corrisponde ad un operatore logico AND.

Per riassumere il comportamento di una porta logica, si ricorre alle cosiddette tavole di veridicità. Per la porta AND, la tavola è in tabella a destra del circuito. Si osservi, per esempio, che quando  $A=1$  e  $B=1$  (entrambi gli interruttori abbassati), allora la lampadina (indicata con C) è accesa e quindi  $C=1$ .

Ora, come secondo passo, esaminiamo un altro circuito: in questo caso, è facile riconoscere che la lampadina si accenderà solo se almeno uno degli



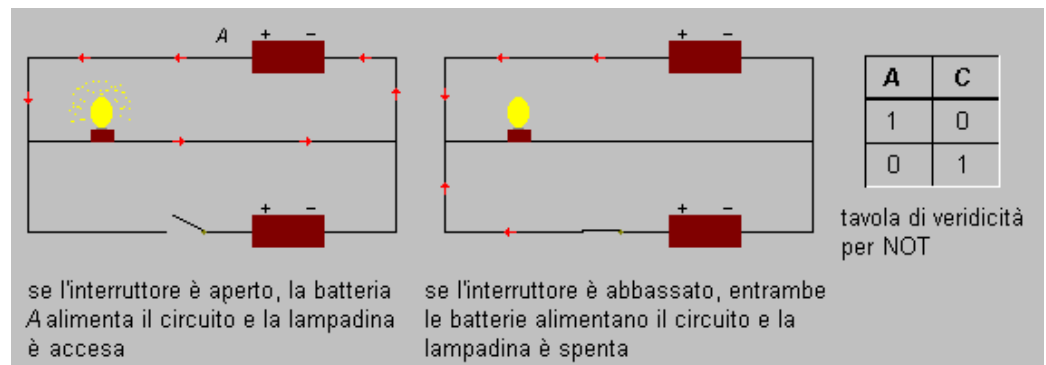
interruttori A e B verrà abbassato in modo da chiudere il contatto elettrico. Questo circuito, corrisponde all'operatore logico OR.

La sua rappresentazione secondo una normativa standardizzata, è rappresentata con il simbolo sottoindicato. A destra, è riportata la corrispondente tavola di veridicità. Si noti come sia sufficiente che un solo interruttore sia abbassato per accendere la lampadina.

Oltre alle porte AND e OR, c'è la porta NOT, capace di invertire il segnale in ingresso: se vale 1, diventa 0 e viceversa.



Il corrispondente circuito, comprende due alimentatori con polarità opposta.



Quando il circuito inferiore è aperto (0), la batteria A alimenta il circuito superiore e la lampadina è accesa; quando il circuito inferiore è chiuso, la seconda batteria fornisce una corrente uguale e opposta che impedisce il passaggio di corrente per cui la lampadina si spegne. La tavola di veridicità corrispondente a questo circuito, comprende un solo ingresso. Si osservi che quando l'interruttore è alzato ( $A=0$ ), la lampadina è accesa ( $C=1$ ) e viceversa.

## In sintesi

Riassumiamo nelle tabelle seguenti i risultati che restituiscono gli operatori logici.

### AND – Congiunzione

falso AND falso	risultato falso
falso AND vero	risultato falso
vero AND falso	risultato falso
vero AND vero	risultato vero

AND (in Action script &&) restituisce un valore vero "se e solamente se gli altri due valori sono veri", questo vuol dire che anche in una successione di più operazioni AND basta che un valore sia falso ed anche il risultato lo sarà.

### OR - Disgiunzione

falso OR falso	risultato falso
falso OR vero	risultato vero
vero OR falso	risultato vero
vero OR vero	risultato vero

L'OR (in Action Script ||) restituisce un valore vero "se e solamente se almeno uno dei due valori risulta vero"; in poche parole anche in una successione di più operazioni OR basta che un valore sia vero ed anche il risultato lo sarà.

## NOT – Negazione

NOT **falso**

risultato **vero**

NOT **vero**

risultato **falso**

Il NOT (in Action Script !) si riduce ad una semplice operazione di negazione del valore acquisito.

---

## Gli elementi del linguaggio

Prendiamo ora in esame la grammatica di base di ogni linguaggio di programmazione, i mattoni con cui costruiremo i programmi.

Come punto di riferimento prenderemo il linguaggio con cui avremo a che fare in questo corso: ActionScript il linguaggio utilizzato da FLASH e che deriva da JavaScript, ma la maggior parte delle cose di cui parleremo varranno, in generale, per tutti i linguaggi.

I termini utilizzati all'interno di un linguaggio si suddividono in Parole chiave, Operatori e separatori, Letterali (o Costanti) e Identificatori definiti dal programmatore (nomi di classi, di variabili, di funzioni). Queste unità semantiche di base del linguaggio le definiremo token. Ora le vedremo in dettaglio, ma prima di iniziare è necessaria una premessa importante. Alcuni linguaggi sono case sensitive altri no. In ActionScript e Java (ma anche, ad esempio, in C e in JavaScript) le parole While, while e WHILE sono per il compilatore parole differenti e solo while è un'istruzione del linguaggio. In altri linguaggi (Pascal, Basic, ecc.) le parole hanno lo stesso significato indipendentemente dal fatto che i caratteri che le compongono siano maiuscoli o minuscoli.

È necessario, quindi, fare molta attenzione. Come vedremo, molti errori derivano, banalmente, dall'errata scrittura di parole chiave o identificatori. In ActionScript la cosa è ulteriormente complicata dal fatto che fino a Flash 6 ActionScript NON era case sensitive, dalla versione 7 di Flash è, invece, diventato case sensitive. Codice scritto per Flash 6 potrebbe quindi non girare in Flash 7 solo perché qualche parola chiave o qualche nome di variabile non è stato scritto con le maiuscole al posto giusto.

### Parole chiave

Le parole chiave sono le parole, o meglio i termini composti da caratteri alfanumerici, riservate al linguaggio di programmazione. Il creatore del linguaggio di programmazione stabilisce a priori quali termini riservare e quale sarà la loro funzione, il compito del programmatore è quello di impararle ed usarle in maniera appropriata. Fortunatamente l'uso improprio

---

di tali termini viene generalmente rilevato durante la fase di compilazione di un programma.

Qui di seguito vengono riportate, come esempio, le parole chiave di ActionScript 2.0.

```
add, and, break, case, catch, class, continue, default,
delete, do, dynamic, else, eq, extends, finally, for,
function, ge, get, gt, if, ifFrameLoaded, implements, import,
in, instanceof, interface, intrinsic, le, lt, ne, new, not,
on, onClipEvent, or, private, public, return, set, static,
switch, tellTarget, this, throw, try, typeof, var, void,
while, with
```

Ecco, invece, ad esempio, quelle del linguaggio JAVA:

```
abstract, assert, boolean, break, byte, case, catch, char,
class, const, continue, default, do, double, else, enum,
extends, final, finally, float, for, goto, if, implements,
import, instanceof, int, interface, long, native, new,
package, private, protected, public, return, short, static,
strictfp, super, switch, synchronized, this, throw, throws,
transient, try, void, volatile, while
```

Come potete vedere molto sono presenti in entrambi i linguaggi.

Le parole chiave sono parole riservate, non possono essere cioè usate per come nomi definiti dal programmatore. La ragione è piuttosto semplice da capire: se assegno ad una funzione il nome new o ad una variabile il nome this, il compilatore non sarebbe in grado di distinguere tra l'uso di quella parola come elemento fondamentale del linguaggio di programmazione o semplice nome da utilizzare per memorizzare dati.

## Operatori e separatori

In combinazione con le parole chiave vengono usati alcuni token composti di uno o più caratteri speciali che servono a controllare il flusso delle operazioni che dobbiamo eseguire o a costruire espressioni, questi token vengono chiamati **operatori**. Questi sono gli operatori usati sia da ActionScript che da JAVA:

```
++ ! != !== % %= & && &= ( ) - * *= , . ?: / // /*
/= [] ^ ^= {} | || |= ~ + += < << <=< <= <> = -=
== === > >= >> >=> >>> >>=>
```

I **separatori** invece sono simboli di interpunzione che permettono di chiudere un'istruzione o di raggruppare degli elementi. Il separatore principale è lo spazio che separa i token tra di loro quando non ci sono altri separatori. Gli altri separatori sono:

```
( ) { } , ;
```

---

Alcuni simboli possono essere usati indistintamente come separatori o come operatori (ad esempio la parentesi), l'utilizzo nella maniera più appropriata si evince dal contesto.

Gli operatori hanno delle proprietà che ne caratterizzano l'uso semplice o composito, tali proprietà permettono infatti di interpretare in modo unico il significato di un'espressione.

Ecco le proprietà degli operatori:

- **POSIZIONE**  
La posizione di un operatore rispetto ai suoi operandi (detti anche argomenti) è molto importante. Un operatore si dice **prefisso** se viene posto prima degli operandi, **postfisso** se viene posto dopo e **infixo** se si trova tra gli operandi.
- **ARIETÀ**  
L'arietà è il numero di argomenti di un operatore; ad esempio il simbolo "+" (che incrementa di uno) ha un solo argomento, il simbolo "+" e le parentesi hanno, invece, due argomenti, l'operatore ":" (operatore condizionale, simile all'IF) è l'unico ad avere tre argomenti.
- **PRECEDENZA (O PRIORITÀ)**  
Per precedenza si intende l'ordine con il quale verranno eseguiti gli operatori. L'esempio classico è quello dell'espressione **4+7\*5** in cui non sappiamo se bisogna eseguire prima l'operazione di somma o di moltiplicazione; premesso che l'operatore di moltiplicazione ha una priorità maggiore rispetto all'addizione, il modo corretto di interpretare l'espressione sarà **4+(7\*5)**.
- **ASSOCITIVITÀ**  
Questa proprietà ci viene in aiuto quando eseguiamo due o più operatori aventi stessa precedenza. Un operatore può essere associativo a **sinistra** oppure associativo a **destra**, questo vuol dire che iniziamo ad eseguire gli operatori partendo rispettivamente da sinistra o da destra.

## Letterali (o costanti)

Le costanti (o letterali) sono quantità note a priori il cui valore non dipende dai dati d'ingresso e non cambia durante l'esecuzione del programma. Le costanti possono essere numeri, stringhe di caratteri, valori di verità.

### Costanti numeriche

I token che indicano costanti numeriche iniziano sempre con un carattere numerico: il fatto che un token inizi con un numero basterà ad indicare al

compilatore che si tratta di una costante numerica. Se il compilatore non potrà valutare quel token come numero segnalerà un errore.

Il segno che separa la parte intera di un numero dalla parte decimale è il punto.

È possibile inserire numeri in formato decimale, binario, ottale o esadecimale. Per scrivere una costante numerica in formato esadecimale oltre ai numeri e al punto dovremo usare anche delle prime sei lettere dell'alfabeto. Per segnalare al compilatore che un numero non è decimale si fa precedere il numero da un prefisso. Per i numeri esadecimali (gli unici che ci capiterà di usare) questo prefisso è "0x" (che non a caso inizia con un numero, il numero zero).

Un'ultima nota. Da quanto abbiamo appena detto deriva che tutti gli altri token (parole chiave e nomi) NON possono iniziare con un numero.

Alcuni esempi di costanti numeriche:

```
1
2433
1000000000
3.14
.3333333333
0.5
2345.675
0xFF0088
0x5500ff
0xff.00aa
```

## Costanti stringa

Una stringa è una sequenza di caratteri UNICODE ed permette di rappresentare testi. Un costante stringa è una sequenza (anche vuota) di caratteri racchiusi tra apici singoli o apici doppi.

Racchiudendo la sequenza tra apici singoli occorre prestare attenzione all'uso degli apostrofi. Nel set di caratteri UNICODE, l'apostrofo e l'apice singolo coincidono. Vedremo più avanti quali problemi possono sorgere e come evitarli.

Per inserire ritorni a capo, tabulazioni, particolari caratteri o informazioni di formattazione si utilizzano speciali sequenze di caratteri dette sequenze di escape. Una sequenza di escape è formata da un carattere preceduto dal simbolo "\" (backslash). La sequenza di escape inserisce un carattere che non sarebbe altrimenti rappresentabile in un letterale stringa.

Le principali sequenze di escape sono:

```
\n: nuova riga;
\r: ritorno a capo;
```

---

```
\t: tabulazione orizzontale;
\' : apostrofo (o apice singolo);
\" : doppio apice;
\\ : backslash(essendo un carattere speciale deve essere
inserito con una sequenza di escape).
```

Ecco alcuni esempi di letterali stringa corretti:

```
// Stringa racchiusa da apici singoli
'Ciao a tutti'
// Stringa racchiusa tra apici doppi
"Ciao"
/* La sequenza di escape risolve l'ambiguità tra l'apostrofo
inserito nella stringa e gli apici singoli che la
racchiudono */
'Questo è l\'esempio corretto'
/* In questo caso non c'è ambiguità perché la stringa è
racchiusa tra doppi apici */
"Anche questo è l'esempio corretto"
/* Per inserire un ritorno a capo si usano le sequenze
di escape */
"Questa è una stringa valida\rdi due righe"
```

I seguenti sono invece esempi di letterali stringa non corretti:

```
/* Nel caso seguente l'apostrofo è interpretato come
l'apice di chiusura della stringa */
'Questo è l'esempio errato'
/* Il tipo di apici che inizia la stringa e quello che
la chiude devono essere uguali */
"Questa stringa non è valida"
// I letterali stringa devono comparire su un'unica riga
"Anche questa stringa
non è valida"
```

## Costanti booleane

Le costanti booleane, poiché rappresenta valori logici, possono avere solo due valori: vero (rappresentato dal letterale true) e falso (rappresentato dal letterale false).

## Array e oggetti

In ActionScript e in altri linguaggi è possibile inserire anche costanti di tipo Array e costanti di tipo Object. Chiariremo meglio in seguito cosa sono array e oggetti. Per ora basti dire che entrambi consentono di organizzare più valori nello stesso contenitore: l'array è un elenco di elementi ai cui valori si può accedere per indice (che è il numero che rappresenta la posizione dell'elemento nell'array) mentre gli oggetti rappresentano un gruppo di proprietà il cui valore è associato a una chiave (il nome della proprietà).

Il letterale Array è costituito da una serie di elementi separati da virgole compresa tra due parentesi quadre:

```
// array che contiene i mesi dell'anno  
["January", "February", "March", "April"];
```

Il letterale Object è invece compreso tra parentesi graffe ed è costituito da una serie di copie "chiave: valore" separate da virgole:

```
//record di una rubrica telefonica in formato Object  
{name:"Irving",age:32,phone:"555-1234"};
```

## Altre costanti

I linguaggi normalmente definiscono anche altre costanti. Alcune rappresentano valori speciali che non possono essere rappresentati che da un valore simbolico (come abbiamo visto per true e false) altre sono valori rappresentati da un letterale per comodità, ma possono essere anche rappresentate, a seconda dei casi, come stringe o come valori numerici.

Per quanto riguarda il primo gruppo ActionScript definisce Infinity, -Infinity, undefined, null e NaN: Infinity e -Infinity rappresentano l'infinito positivo e l'infinito negativo, undefined è il valore speciale che contiene una variabile prima di essere usata, null è un valore che indica che una variabile non contiene nulla (vedremo in pratica la differenza tra questi due valori), NaN (significa Not a Number, attenzione all'uso corretto di maiuscole e minuscole!) è il valore che restituisce un'espressione numerica quando il calcolo non può essere eseguito (normalmente perché uno degli operandi non è un numero). NaN, null e undefined sono definiti da molti altri linguaggi.

Il secondo gruppo è molto vario e cambia molto da linguaggio a linguaggio. ActionScript definisce newline che equivale a "\r" (ritorno a capo) come costante globale. Molte altre costanti vengono definite come proprietà di classi predefinite. La classe Math, ad esempio, fornisce una serie di utili costanti numeriche: Math.PI (3.14159265358979 o  $\pi$ ), Math.SQRT2 (1.4142135623731, radice quadrata di due), ecc.

## Identificatori

Un identificatore è un nome definito dal programmatore. Gli identificatori si usano per dare nomi alle variabili, alle funzioni e ai tipi di dati (o classi). Parleremo in dettaglio di questi elementi nei capitoli seguenti. Qui basterà ricordare le regole a cui attenersi nella scelta degli identificatori:

- il primo carattere deve essere una lettera o il simbolo "\_" (ricordiamo che nel caso la prima lettera fosse un numero il compilatore tenterebbe di interpretare il token come costante numerica);
  - i caratteri successivi possono essere lettere, numeri o "\_".
-



Gli identificatori non possono inoltre coincidere con le parole riservate del linguaggio.



## Variabili e Tipi di Dati

### Variabili

Adesso è fondamentale definire il concetto di variabile, visto che nelle prossime lezioni faremo largo uso di questo termine.

Pensiamo, ad esempio, a quando salviamo un numero di telefono del nostro amico Mario sul cellulare; se vogliamo chiamare il nostro amico, basterà inserire il suo nome (Mario, nome della variabile) ed il cellulare comporrà automaticamente il numero di telefono (valore della variabile). Se per qualche ragione Mario cambierà numero di telefono, modificherò il contenuto della mia rubrica (cambierò il valore della variabile). In questa maniera senza modificare le mie abitudini (inserirò sempre Mario) il mio cellulare comporrà il nuovo numero.

Si può vedere quindi che una variabile è composta da due elementi: il suo nome e il suo valore; come ho visto nell'esempio del cellulare in un programma posso usare i nomi delle variabili al posto dei valori che rappresentano. Questo ha due vantaggi. Il primo è la possibilità di usare simboli mnemonici al posto di numeri e stringhe di grande entità o difficili da ricordare (mi è più comodi scrivere Mario che il suo numero di telefono che potrei non ricordare). Il secondo è la possibilità di usare il nome della variabile al posto del suo valore per eseguirvi sopra delle operazioni, con la possibilità, in seguito, di modificare il valore come e quante volte vogliamo, in altri termini di generalizzare un'elaborazione.

Facciamo un esempio. Proviamo a scrivere un programma che calcola il quadrato di un numero inserito dall'utente e lo mostra sul video. Per il momento non useremo un linguaggio specifico. Useremo uno pseudolinguaggio inventato da noi. Come vedete "A" può assumere un valore a piacere. In questa maniera possiamo generalizzare l'operazione.

```
scrivi sullo schermo "Ciao Inserisci un numero";  
A = -numero inserito da tastiera-;  
B = A * A;  
scrivi sullo schermo "Il quadrato di " A " è " B;  
/* Anche B è una variabile e viene usata per registrare il  
risultato finale */
```

Se inseriamo come numero il valore "4", sullo schermo verrà scritto "Il quadrato di 4 è 16", se scriviamo 2 sullo schermo verrà scritto "Il quadrato di 2 è 8" e così via...

## Tipi

Le variabili possono contenere vari tipi di dati. Un tipo di dato o, più semplicemente un tipo definisce come le informazioni verranno codificate per essere elaborate o semplicemente memorizzate.

Prima di usare una variabile è necessario dichiararla. La dichiarazione è un comando che comunica al compilatore che un determinato letterale è il nome di una variabile e che quella variabile conterrà un determinato tipo di dati. La dichiarazione consente al compilatore di controllare alcuni errori e di segnalarceli: ad esempio se sbagliamo a scrivere il nome di una variabile o se assegniamo alla variabile un valore di un tipo diverso da quello che avevamo dichiarato il compilatore ce lo segnala, facilitandoci la fase di debugging.

In linea di massima esistono due categorie di tipi di dati. I tipi primitivi e i tipi derivati.

### Tipi primitivi

I tipi primitivi sono quelli fissati dalle specifiche del linguaggio.

La prima caratteristica dei tipi primitivi è che possiamo manipolarli utilizzando degli operatori.

Quando dobbiamo operare su un dato, lo dobbiamo mettere in una variabile. Una variabile è come una scatola che ci consente di maneggiare e di memorizzare i dati. Un'altra caratteristica dei tipi primitivi è che la scatola li contiene completamente.

Sembra ovvio, ma , come vedremo, non lo è affatto. Se pensiamo alle variabili come scatole possiamo dire che se contengono tipi primitivi aprendole troveremmo direttamente i dati, se invece contengono dati di un tipo derivato aprendole troveremmo il numero di scaffale che i dati occupano nella memoria del computer. Nel primo caso la variabile conterrà direttamente il dato. Nel secondo caso conterrà il puntatore al dato ossia l'indirizzo di memoria in cui il dato è stato collocato. Come vedremo in seguito questa differenza non è affatto secondaria.

I tipi primitivi definiti in ActionScript sono :

#### Boolean

Il tipo di dati Boolean può avere due valori: true e false. Nessun altro valore è consentito per le variabili di questo tipo. Il valore predefinito di una variabile booleana dichiarata ma non inizializzata è false.

---

int e uint

Il tipo di dati int memorizza un numero intero utilizzando 32 bit (4 byte). Il valore può andare da -2,147,483,648 ( $-2^{31}$ ) a 2,147,483,647 ( $2^{31} - 1$ ) inclusi.

Il tipo uint memorizza un numero intero positivo utilizzando 32 bit (4 byte). Il valore può andare da da 0 a 4,294,967,295 ( $2^{32} - 1$ ) inclusi.

Le precedenti versioni di ActionScript offrivano solo il tipo Number per rappresentare sia i numeri interi che i numeri decimali. Se nel vostro programma usate numeri interi, usando variabili di tipo int e uint aumenterete sensibilmente le prestazioni. Per memorizzare valori inferiori o superiori a quelli indicate bisogna utilizzare il Tipo Number. Il valore di default per le variabili di tipo Int e Uint è 0.

Number

Questo tipo di dati può rappresentare numeri interi, numeri interi senza segno e numeri a virgola mobile utilizzando 64 bit (8 byte). La conversione tra numero intero e numero decimale è stomatica: per utilizzare un numero decimale, bisogna includere il punto decimale (Ad esempio: 123.0 – Nota che Action Script utilizza la notazione anglosassone dove il segno per separare la parte intera dalla parte decimale di un numero è il punto). Se il numero è impostato come intero intero i risultati delle operazioni vengono arrotondate al numero intero più vicino.

Il valore Massimo e il valore minimo che il tipo Number è in grado di contenere sono memorizzati nelle proprietà statiche MAX\_VALUE e MIN\_VALUE della calsse Number.

```
Number.MAX_VALUE == 1.79769313486231e+308  
Number.MIN_VALUE == 4.940656458412467e-324
```

La notazione qui utilizzata e quella scientifica (o esponenziale) ed equivale (nel primo caso) a  $1.79769313486231 * 10^{308}$ , cioè 179,769,313,486,231 seguito da 294 zeri.

Questa possibilità di rappresentare numeri estremamente grandi ed estremamente piccoli viene pagata in termini di precisione nei calcoli. Si avrà una buona approssimazione per i calcoli su numeri relativamente vicini allo zero, mentre calcoli su numeri moilto grandi o molto piccoli saranno molto approssimati.

Quando viene utilizzato per memorizzare i numeri interi il tipo Number è in grado di gestire valori da -9,007,199,254,740,992 ( $-2^{53}$ ) a 9,007,199,254,740,992 ( $2^{53}$ ) compresi.

Il valore di default di una variabile Number è NaN.

## String

Il tipo di dati String rappresenta una sequenza di caratteri a 16 bit che può includere lettere, numeri e segni di punteggiatura. Le stringhe vengono memorizzate come caratteri Unicode, utilizzando il formato UTF-16.

Un'operazione su un valore String restituisce una nuova istanza della stringa.

## Null

Il tipo di dati Null contiene un unico valore: `null`; `null` è il valore che contengono le variabili di tipo String e tutte le variabili complesse quando sono state create, ma non è stato loro assegnato alcun valore.

## void

Il tipo di dati void contiene un unico valore: `undefined`. Viene usato solo per designare le funzioni che non ritornano alcun valore.

Ecco alcuni esempi di dichiarazione di variabili di tipi primitivi:

```
//dichiarazioni di variabili in actionscript

var i:int;           //i conterrà un numero intero
var p:uint;          //p conterrà un numero intero positivo
var n:Number;        //n conterrà un numero reale
var s:String;        //s conterrà una stringa di caratteri
var k:Boolean;       //k conterrà true o false

function pippo():void //dichiaro una funzione che non
                      //ritorna alcun valore

/* dichiarazioni di variabili con assegnazione di un valore
   iniziale */

var b:Number = 1;      //b è il numero intero 1
var n:Number = 234.0;  //n è un numero decimale
var u:uint = 36789345;
var i:int = 53;
var flag:Boolean = true ;
var messaggio:String = "Ciao a tutti";
```

Come potete vedere dagli esempi all'atto della dichiarazione si può assegnare un valore alla variabile (inizializzazione).

Finché alla variabile non viene assegnato un valore essa conterrà il valore speciale `undefined`.

## Tipi derivati o complessi

I tipi primitivi servono a rappresentare dati semplici: un numero, vero o falso, una stringa di caratteri. Per rappresentare dati più complessi (ad esempio un elenco di valori, i dati che compongono un indirizzo, una data, ecc.) ho a disposizione alcuni tipi complessi che il linguaggio mi offre oppure ne posso creare ad hoc.

---

Come anticipavamo la differenza fondamentale tra i tipi primitivi e i tipi derivati è che nel primo caso le variabili contengono direttamente i dati (vero, falso, un numero, una stringa) nel secondo caso la variabile contiene il *puntatore* cioè il numero della casella, l'indirizzo in cui il dato è memorizzato sul computer.

---





## La programmazione Condizionale

Il modo in cui strutturiamo un ragionamento, generalmente, segue la logica procedurale, cioè avendo delle operazioni da fare le eseguiamo una dopo l'altra in sequenza dalla prima all'ultima; anche nella programmazione avviene la stessa cosa, infatti quando scriviamo un programma dobbiamo pensare a creare una schermata introduttiva, chiedere i dati all'utente e poi fornirgli il risultato. Senza entrare nel dettaglio, perché questa guida è per principianti cercheremo ora di spiegare i costrutti più usati nei linguaggi di programmazione per "incanalare" il flusso del nostro programma verso la soluzione (vedere la lezione che parla dei Diagrammi di Flusso); questi strumenti sono chiamati Istruzioni Condizionali perchè permettono di eseguire del codice a seconda che una condizione sia vera o falsa.

Il costrutto principale, usato universalmente, è l'**if**, il quale permette di porre una condizione ed eseguire delle scelte in base ai dati fornitigli. Qui di seguito cerchiamo di spiegarlo sempre utilizzando uno pseudolinguaggio:

```
istruzione-if :  
    if ( espressione ) istruzione  
    if ( espressione ) istruzione else istruzione
```

L'espressione che compare dopo la parola chiave **if** deve essere di tipo logico, se la condizione risulta vera viene eseguita l'istruzione subito seguente; nel secondo caso, invece, se la condizione risulta falsa si esegue l'istruzione seguente, altrimenti si esegue l'istruzione subito dopo la parola chiave **else**. Poiché le istruzioni all'interno dell'**if** possono essere anche più di una è opportuno, per non creare confusione, inserirle all'interno delle parentesi graffe '{' e '}', anche se generalmente un else si riferisce sempre all'if che gli sta più vicino. Per più scelte invece si può usare l'**else if** che permette di porre una condizione anche per le alternative, lasciando ovviamente la possibilità di mettere l'**else** (senza condizioni) in posizione finale.

---

Qui riportiamo un semplice programma, sempre scritto in pseudocodice, che utilizza il costrutto if e le sue varianti finora esposte:

```
intero A = 50;
scrivi sullo schermo "Inserisci un numero";
intero B = -numero inserito da tastiera-;
if (B minore di A) {
    scrivi sullo schermo "Il numero inserito è minore di
    cinquanta";
} else if (B maggiore di A) {
    scrivi sullo schermo "Il numero inserito è maggiore di
    cinquanta";
} else if (B uguale a A) {
    scrivi sullo schermo "Il numero inserito è cinquanta";
} else {
    //poiché B non è minore, né maggiore né uguale a A
    // A non è un numero
    scrivi sullo schermo "Inserisci un numero";
}
```


Questo programma aspetta che l'utente inserisca un numero tramite la tastiera (che viene memorizzato nella variabile B) e poi lo confronta con il valore predefinito (50 che viene memorizzato nella variabile A), a seconda che il valore inserito sia più grande, più piccolo o uguale a tale numero, il programma visualizza sullo schermo una frase che ci dice il risultato di tale confronto.

### Esempio in Action Script

Per realizzare questo programma in in Flash con Action Script come prima cosa dovremo creare sullo stage tre campi di testo. Per fare questo utilizzeremo l'apposito tool presente nella *palette strumenti* e utilizzando la *palette proprietà* definiremo le caratteristiche dei campi.

Il primo campo sarà di tipo *testo statico* e qui inseriremo il testo *"Inserisci un numero"*. Il secondo sarà di tipo *testo di input* e servirà a raccogliere il numero inserito dall'utente: come font del campo sceglieremo *\_sans* e gli daremo il nome *input\_txt*. Il terzo sarà un campo di tipo *testo dinamico* e servirà a mostrare sullo schermo la frase che ci segnalerà il risultato del nostro confronto: come font del campo sceglieremo *\_sans* e gli daremo il nome *messaggio\_txt*.

A questo punto premendo F5 aggiungeremo un fotogramma al nostro filmato in modo che sia composto da almeno due fotogrammi e aggiungeremo un livello che chiameremo *Azioni*.

Selezioniamo il primo frame sul livello azioni e apriamo la finestra delle azione cliccando sulla palette delle proprietà il simbolo . Sulla barra della finestra delle azione dovrà essere scritto: *Azioni - Fotogramma* ad indicare che stiamo modificando le azioni per quel fotogramma.

Inseriamo il codice seguente:

```
var A:Number = 50;
var B:Number = input_txt.text;
if (B < A) {
    messaggio_txt.text = "Il numero inserito è minore di
cinquanta";
} else if (B > A) {
    messaggio_txt.text = "Il numero inserito è maggiore di
cinquanta";
} else if (B == A) {
    messaggio_txt.text = "Il numero inserito è cinquanta";
} else { //B non è un numero
    messaggio_txt.text = "Inserisci un numero!!!";
}
```

Scegliamo sul menu *controllo* e quindi *prova filmato*. È possibile trovare il filmato che contiene questo codice nella cartella Esempi del CD:

*ProgrammazioneCondizionale fla.*

NB: Il filmato deve essere composto di almeno due fotogrammi perché solo in questo caso gira indefinitamente in *loop*. Ogni volta che si torna al primo fotogramma, il codice che controlla quanto inserito dall'utente viene ripetuto e il contenuto dei campi aggiornato.

---



## La programmazione Iterativa

Fino ad ora abbiamo visto che la modalità generale di programmazione è quella procedurale: viene eseguita un'istruzione dopo l'altra fino a che non si incontra l'istruzione di fine programma. Esiste anche un'altra logica, quella iterativa; un'istruzione (o una serie di istruzioni) vengono eseguite continuamente, fino a quando non sopraggiungono delle condizioni che fanno terminare tale computazione. Facciamo l'esempio più semplice, voler stampare a video per mille volte la solita frase; la programmazione procedurale suggerisce di scrivere mille volte il codice, mentre quella iterativa permette di scrivere il comando **una ed una sola volta** e poi ripeterlo per mille volte, dopo le quali una struttura di controllo adeguata (ad esempio un contatore da uno a mille) termine l'esecuzione del ciclo.

### While, do e for

I tre costrutti comunemente usati sono il WHILE, il DO ed il FOR che fanno praticamente la stessa con modalità leggermente diverse.

L'istruzione WHILE viene schematizzata come segue:

```
istruzione-while :  
    while ( condizione ) blocco istruzioni
```

Con questa istruzione viene prima valutata la condizione racchiusa tra le parentesi tonde, se l'espressione risulta vera viene eseguita l'istruzione all'interno del while e il while viene ripetuto, altrimenti si esce dal ciclo del while e si procede con il resto del programma.

Il DO può essere considerato una variante dell'istruzione while ed è strutturato nella maniera seguente:

```
istruzione-do :  
    do blocco istruzioni while ( condizione )
```

---

Prima di tutto viene eseguita l'istruzione racchiusa tra **DO** e **WHILE** (quindi si esegue almeno una volta), poi si verifica il risultato dell'espressione, se è vero si riesegue il DO, altrimenti si continua con l'esecuzione del resto del programma. L'unica differenza rispetto al costrutto precedente è che il blocco di istruzioni da iterare viene eseguito almeno una volta.

Il FOR, schematizzato qui sotto, inizializza una variabile, pone una condizione (che deve essere vera o falsa) e poi modifica (normalmente incrementa o decrementa) la variabile iniziale.

```
istruzione-for :  
  for (inizializzazione; condizione; modifica variabile )  
    blocco istruzioni;
```

Volendo schematizzare questo ciclo utilizzando l'istruzione WHILE, ecco la struttura che vi si presenta:

```
inizializzazione variabile;  
while ( condizione ){  
  istruzione1;  
  istruzione2;  
  ....  
  modifica variabile;  
}
```

Come vedete le strutture for e while in molti casi sono intercambiabili. La scelta per l'una o per l'altra dipende molto dello stile del programmatore e dalla leggibilità del programma.

## Un esempio: la successione di Fibonacci

Leonardo Fibonacci fu un matematico Pisano vissuto tra 1170 e il 1250. Tra il 1202 e 1220 pubblicò la sua opera di quindici capitoli *Liber Abaci*, tramite la quale introdusse per la prima volta in Europa le nove cifre (da lui chiamate *indiane*), assieme al segno 0, "che in arabo è chiamato zefiro" (cap. I), con confronti con il sistema romano, presentò criteri di divisibilità, regole di calcolo di radicali quadratici e cubici ed altro. Introdusse con poco successo la barretta delle frazioni (nota al mondo arabo prima di lui) (cap. II-IV), ma Fibonacci è soprattutto famoso per la sequenza di numeri interi che da lui prende il nome.

La **successione di Fibonacci** è una sequenza di numeri interi naturali definibile assegnando i valori dei due primi termini,  $F_0 := 0$  ed  $F_1 := 1$ , e chiedendo che per ogni successivo sia  $F_n := F_{n-1} + F_{n-2}$ . Il termine  $F_0$  viene

---

aggiunto nel caso si voglia fare iniziare la successione con 0; storicamente il primo termine della successione è  $F_1 = 1$ .

In altri termini dati due numeri interi iniziali (0 e 1 o 1 e 1) i numeri successivi della sequenza sono dati dalla somma di due numeri immediatamente precedenti.

La successione di Fibonacci possiede moltissime proprietà di grande interesse. Certamente la proprietà principale, e maggiormente utile nelle varie scienze, è quella per cui il rapporto  $F_n / F_{n-1}$  al tendere di  $n$  all'infinito tende al numero algebrico irrazionale chiamato sezione aurea o numero di Fidia

$$\phi = \frac{1 + \sqrt{5}}{2} = 1,6180339887$$

Curioso anche notare come, salendo via via per la sequenza, questo rapporto risulti alternatamente maggiore e minore della costante limite. Naturalmente il rapporto tra un numero di Fibonacci e il suo successivo tende al reciproco della sezione aurea.

$$1/\phi = 0,6180339887...$$

Conviene anche ricordare che:

$$a) \quad \phi - 1 = 1/\phi$$

$$b) \quad 1 - \phi = -1/\phi = \frac{1 - \sqrt{5}}{2}$$

in accordo con la definizione di sezione aurea come il numero positivo tale che

$$\phi = 1 + 1/\phi$$

equazione che, quando vincolata alla condizione  $\phi > 0$ , ammette l'unica soluzione:

$$\phi = \frac{1 + \sqrt{5}}{2}$$

Si noti poi come l'inverso del reciproco del numero di Fidia  $-1/\phi$  che compare nella b) costituisca la seconda soluzione, a segno negativo, dell'equazione algebrica riportata nella definizione. Esso espone proprietà altrettanto interessanti di quelle del suo omologo positivo.

Ragionamenti analoghi possono essere applicati per ottenere altri rapporti irrazionali costanti; per esempio dividendo ogni numero per il secondo successivo si ottiene 0.382 e dividendo ogni numero per il terzo successivo si ottiene 0.236, mentre dividendo ogni numero per il secondo precedente si ottiene 2.618 e dividendo ogni numero per il terzo precedente si ottiene 4.236.

Questi sei rapporti (0,236, 0,382, 0,618, 1, 1,618, 2,618, 4,236) sono inoltre molto utilizzati nella teoria delle onde di Elliott, argomento molto importante dell'analisi tecnica dei mercati finanziari

Altra proprietà interessante è che se un qualsiasi numero della successione è elevato al quadrato, questo è uguale al prodotto tra il numero che lo precede e quello che lo segue, aumentato o diminuito di una unità: per esempio  $5^2 = 25 = 3 * 8 + 1$  oppure  $3^2 = 9 = 2 * 5 - 1$ .

Si trova anche che l' $n$ -simo numero di Fibonacci si può esprimere con la formula:

$$F(n) = \frac{\phi^n}{\sqrt{5}} - \frac{(1-\phi)^n}{\sqrt{5}} = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}$$

Tale formula, di rara bellezza ed eleganza, porta il nome di Jacques Binet, che la ricavò nel 1843; tuttavia essa era già nota nel XVIII secolo ad Eulero, Abraham De Moivre e Daniel Bernoulli.

Ecco un semplice programma per scrivere sullo schermo  $N$  numeri della sequenza di Fibonacci.

Prima scriveremo il programma in uno pseudolinguaggio, cercando cioè di astrarci da uno specifico linguaggio di programmazione. In seguito vedremo

di adattare il nostro programma ad Action Script e alle caratteristiche di FLASH.

```
//scrive la serie di Fibonacci
intero n1 = 0; //contiene il numero n-2 della serie
intero n2 = 1; // contiene il numero n-1 della serie
intero n3; //numero di Fibonacci che viene calcolato
stringa elenco = ""; //Stringa di testo che conterrà l'elenco
//dei numeri di Fibonacci
scrivi sullo schermo "Inserisci un numero maggiore di 2";
intero A = - numero inserito dall'utente -

if (A maggiore o uguale 2) {
    //se il numero inserito è maggiore di due elenco conterrà
    //all'inizio i primi due numeri della serie
    elenco = "0, 1, ";
}

for (intero i = 0; i minore di (A-2); incrementa i) {
    //il numero di viene calcolato sommando i due numeri
    //precedenti nella serie
    n3 = n2 + n1;
    //in n1 e n2 vengono ora memorizzati gli ultimi due numeri
    //della serie
    n1 = n2;
    n2 = n3;
    //a elenco viene aggiunto il numero di Fibonacci così
    // ottenuto più una virgola e uno spazio
    elenco = elenco + n3 + ", ";
}
// il ciclo continua fino a che non sono stati creati tutti
// numeri richiesti dell'utente.

// la stringa così ottenute viene scritta sullo schermo
scrivi sullo schermo elenco;
```

## La successione di Fibonacci in Action Script


Per realizzare questo programma in Flash con Action Script come prima cosa dovremo creare sullo stage tre campi di testo. Per fare questo utilizzeremo l'apposito tool presente nella *palette strumenti* e utilizzando la *palette proprietà* definiremo le caratteristiche dei campi.

Il primo campo sarà di tipo *testo statico* e qui inseriremo il testo *"Inserisci un numero maggiore di 2"*. Il secondo sarà di tipo *testo di input* e servirà a raccogliere il numero inserito dall'utente: come font del campo sceglieremo *\_sans* e gli daremo il nome *input\_txt*. Il terzo sarà un campo di tipo *testo dinamico* e servirà a mostrare sullo schermo l'output del programma ossia la serie di Fibonacci che avremo generato: come font del campo sceglieremo *\_sans* e gli daremo il nome *messaggio\_txt*.

---



A questo punto premendo F5 aggiungeremo un fotogramma al nostro filmato in modo che sia composto da almeno due fotogrammi e aggiungeremo un livello che chiameremo *Azioni*.

Selezioniamo il primo frame sul livello azioni e apriamo la finestra delle azione cliccando sulla palette delle proprietà il simbolo . Sulla barra della finestra delle azione dovrà essere scritto: *Azioni - Fotogramma* ad indicare che stiamo modificando le azioni per quel fotogramma.

Inseriamo il codice seguente:

```
//scrive la serie di Fibonacci
var n1:Number = 0; //elemento n-2 della serie
var n2:Number = 1; //elemento n-2 della serie
var n3:Number = 0; //numero di Fibonacci calcolare

var elenco:String= ""; //Stringa di testo che conterrà
                        //l'elenco dei numeri di Fibonacci

var A:Number = input_txt.text; //numero inserito dall'utente

if (A >= 2) {
    //se il numero inserito è maggiore di u uguale a 2 elenco
    //conterrà all'inizio i primi due elementi della serie
    elenco = "0, 1, ";
}
for (var i:Number = 0; i < (A-2); i++) {
    //il numero di viene calcolato sommando i due numeri
    //precedenti nella serie
    n3 = n2 + n1;
    //in n1 e n2 vengono ora memorizzati gli ultimi due numeri
    //della serie
    n1 = n2;
    n2 = n3;
    //a elenco viene aggiunto il numero di Fibonacci così
    //ottenuto Più e una virgola e uno spazio
    elenco = elenco + n3 + ", ";
}
// il ciclo continua fino a che non sono stati creati tutti
// i numeri richiesti dell'utente.

// la stringo così ottenute viene scritta nel campo di testo
messaggio_txt.text = elenco;
```

Scegliamo sul menu *controllo* e quindi *prova filmato*. Quando si inserisce un numero nel campo di input viene visualizzate la sequenza di Fibonacci relativa. È possibile trovare il filmato che contiene questo codice nella cartella Esempi del CD il nome del file è *Iterazione\_for fla*.

Lo stesso programma è stato implementato anche usando while al posto di for il file relativo è *Iterazione\_while fla*.

NB: Il filmato deve essere composto di almeno due fotogrammi perché solo in questo caso gira indefinitamente in *loop*. Ogni volta che si torna al primo fotogramma, il codice che controlla quanto inserito dall'utente viene ripetuto e il contenuto dei campi aggiornato di conseguenza.

## Uso dei cicli for..in

ActionScript offre un'ulteriore struttura iterativa il ciclo for..in. Un ciclo for..in consente di eseguire *iterazioni* scorrendo gli elementi secondari di un clip filmato, le proprietà di un oggetto o gli elementi di un array. Gli elementi secondari di un clip filmato sono costituiti da tutto ciò che un clip filmato può contenere: altri clip filmato, funzioni, oggetti e variabili. L'uso più comune di un ciclo for..in è scorrere le proprietà (le coppie valore/chiave) di un oggetto.

È possibile, ad esempio, ricorrere a un ciclo for...in per eseguire iterazioni sulle proprietà di un oggetto generico. Le proprietà degli oggetti non vengono ordinate in base a criteri particolari, ma inserite in ordine imprevedibile:

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj) {
    trace(i + ": " + myObj[i]);
}
```

Il seguente output del codice viene visualizzato nel pannello Output:

x: 20

y: 30

È possibile anche eseguire iterazioni sugli elementi di un array:

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray) {
    trace(myArray[i]);
}
```

## Funzioni e metodi

Quando sviluppiamo un programma capita spesso che alcune operazioni debbano essere ripetute più di una volta. Siccome scrivere tutte le volte le medesime operazioni risulterebbe tedioso quanto inutile ci viene in aiuto il concetto di funzione che altro non è che un programmino (o modulo) a se stante il quale prende in entrata dei valori e restituisce un risultato. Altro vantaggio è quello di poter modificare la struttura della funzione (per ottimizzarla o renderla più veloce) senza per questo dover intaccare il codice che la richiama.

Una funzione ha bisogno di essere **dichiarata** e **definita**; cioè vanno specificati i tipi di ingresso e di uscita sui quali la funzione andrà a compiere le proprie operazioni (DICHIAZIONE) e successivamente dovremo scrivere il "corpo" della funzione vera e propria (DEFINIZIONE).

D'ora in poi ci riferiremo in specifico da ActionScript anche se quando vedremo vale anche per altri linguaggi: innanzi tutto per JavaScript, che ha una sintassi identica, ma, con minime modifiche, anche per JAVA, C++ e C#.

### Funzioni e metodi in ActionScript

I metodi e le funzioni sono blocchi di codice ActionScript riutilizzabili in qualsiasi punto di un file SWF. È possibile creare una funzione nel file FLA o in un file ActionScript esterno e quindi chiamarla da qualunque punto dei documenti. I metodi sono semplicemente funzioni che si trovano all'interno di una definizione di classe ActionScript.

È possibile definire funzioni per eseguire una serie di istruzioni sui valori specificati. Le funzioni possono anche restituire altri valori. Una volta definita una funzione, è possibile chiamarla da qualsiasi linea temporale, inclusa la linea temporale di un file SWF caricato.

Se a una funzione vengono passati valori come parametri, la funzione può eseguire calcoli utilizzando i valori forniti. Ogni funzione presenta caratteristiche proprie e alcune funzioni richiedono il passaggio di un determinato tipo o numero di valori. Se viene passato un numero di parametri superiore a quello richiesto dalla funzione, i valori aggiuntivi

---

vengono ignorati. Se un parametro obbligatorio non viene passato, la funzione assegna ai parametri vuoti il tipo di dati `undefined`.

## Funzioni incorporate

Nel linguaggio `ActionScript` sono incorporate numerosi funzioni che consentono di eseguire determinate attività e di accedere alle informazioni.

Si può trattare di funzioni globali o di funzioni appartenenti ad una classe incorporata nel linguaggio, quale ad esempio `Math` o `MovieClip`, e che, quindi, verranno utilizzate come metodi della classe o dell'oggetto istanza di quella classe. Chiariremo meglio questi concetti quando parleremo delle Classi.

Si può, ad esempio, ottenere il tempo passato da quando un file `SWF` è stato lanciato utilizzando `getTimer()` o il numero di versione di `Flash Player` in cui è caricato il file utilizzando `getVersion()`. Le funzioni appartenenti a un oggetto sono denominate metodi. Quelle che non appartengono a un oggetto sono denominate funzioni di primo livello.

Le funzioni di primo livello sono di facile utilizzo. Per chiamare una funzione, è sufficiente utilizzarne il nome e passare tutti i parametri richiesti. Se, ad esempio, aggiungo il codice `ActionScript` seguente al fotogramma 1 della linea temporale:

```
trace("Hello world!");
```

Quando si prova il file `SWF`, verrà visualizzato `Hello world!` nel pannello `Output`. La funzione `trace`, infatti non fa altro che scrivere un messaggio sulla finestra di `output` e non ritorna alcun valore.

## Funzioni utente

Le funzioni utente sono funzioni create dall'utente e che normalmente consentono di compiere operazioni non previsto dal linguaggio e dalle funzioni incorporate. Esistono due modi per dichiarare e definire le funzioni utente: posso scrivere funzioni con nome o funzioni anonime.

### Scrittura di funzioni con nome

La funzione con nome è quella più comunemente usata. Quando si compila un file `SWF`, le funzioni con nome sono compilate per prime; cioè, è possibile fare riferimento a queste funzioni in qualsiasi punto del codice, a condizione che la funzione sia stata definita nel fotogramma corrente o precedente.

Ad esempio, se una funzione è definita nel fotogramma 2 di una linea temporale, non sarà possibile accedervi dal fotogramma 1 della linea temporale.

La sintassi con cui definisco una funzione con nome è la seguente:

---

```
function nomefunzione (parametro1, parametro2, ....) {
    // Blocco di istruzioni
}
```

Questa porzione di codice comprende le parti seguenti:


- `nomefunzione` è il nome univoco della funzione. Tutti i nomi di funzione in un documento devono essere univoci.
- `parametro1`, `parametro2`, ... uno o più parametri che vengono passati alla funzione. I parametri sono detti anche *argomenti*.
- Blocco di istruzioni contiene tutto il codice ActionScript relativo alla funzione. Questa parte contiene le istruzioni che eseguono le azioni, ovvero il codice che si desidera eseguire. Il commento `// Blocco di istruzioni` è un segnaposto che indica dove deve essere inserito il blocco della funzione.

Proviamo ora a scrivere una funzione utente. La funzione che scriveremo e chiameremo `my_timer` avrà la semplice funzione di scrivere in un campo di testo il tempo passato da quando il filmato è stato aperto. Per farlo utilizzerà la funzione incorporata `getTimer()`.

Come prima cosa dovremo creare sullo stage due campi di testo. Per fare questo utilizzeremo l'apposito tool presente nella *palette strumenti* e utilizzando la *palette proprietà* definiremo le caratteristiche dei campi.

Il primo campo sarà di tipo *testo statico* e qui inseriremo il testo *"Dal via sono trascorsi "*. Il secondo sarà un campo di tipo *testo dinamico* e servirà a mostrare sullo schermo l'output del programma ossia i millisecondi trascorsi: come font del campo sceglieremo `_sans` e gli daremo il nome `messaggio_txt`.

A questo punto aggiungeremo un livello nella linea temporale che chiameremo *Azioni*.

Selezioniamo il primo frame sul livello azioni e apriamo la finestra delle azione cliccando sulla palette delle proprietà il simbolo . Sulla barra della finestra delle azione dovrà essere scritto: *Azioni - Fotogramma* ad indicare che stiamo modificando le azioni per quel fotogramma della linea temporale.

Inseriamo il codice seguente:

```
stop();
//semplice timer che misura il tempo trascorso
//dal lancio del filmato

//definisco la funzione my_timer() che non ritorna alcun
//valore e che aggiorna il contenuto del campo di testo
```

```
//messaggio_txt sulla base di quanto restituito dalla
//funzione incorporata getTimer

function my_timer ():Void {
    //millisecondi trascorsi
    var t:Number = getTimer();
    //secondi = parte intera delle divisione
    var s:Number = Math.floor(t/1000);
    //millesimi = resto della divisione
    var m:Number = t % 1000;

    messaggio_txt.text = s + " secondi e " + m +
        " millesimi.";
}
// la funzione my_timer viene eseguita una prima volta
// appena il filmato viene eseguito

my_timer();

// poi utilizzando il timer del computer la funzione
// my_timer viene eseguita ogni 30 millisecondi

setInterval(my_timer, 30);
```

L'esempio qui riportato si trova nel file FunzioneConNome fla.

Per creare la propria funzione in ActionScript si utilizza l'istruzione `function`. I parametri sono opzionali, ma è necessario includere le parentesi quadre anche se non vengono specificati. Il contenuto tra le parentesi graffe (`{}`) è detto *blocco di funzione*.

Le funzioni sono normalmente scritte sulla linea temporale del filmato principale o all'interno di file ActionScript esterni, ad esempio file di classe.

Un particolare tipo di funzione è la funzione di costruzione di una classe detta *constructor*. In ActionScript queste funzioni usano lo stesso sintassi delle normali funzioni utente. Unica prescrizione è che il nome della funzione deve corrispondere al nome della classe. Vedremo meglio come scrivere funzioni, (cioè metodi) nelle classi, quando parleremo di classi.

Scrittura di funzioni anonime e di callback

Esiste un modo alternativo per definire una funzione: quello di creare una funzione anonima.

Il costrutto tipico con cui definisco una funzione anonima è il seguente:

```
var nomevariabile = function (parametro1, parametro2, ....) {
    // Blocco di istruzioni
}
```

Questa porzione di codice comprende le parti seguenti:

---

- nomevariabile è il nome di una variabile.
- parametro1, parametro2, ... uno o più parametri che vengono passati alla funzione. I parametri sono detti anche *argomenti*.
- Blocco di istruzioni contiene tutto il codice ActionScript relativo alla funzione. Questa parte contiene le istruzioni che eseguono le azioni, ovvero il codice che si desidera eseguire.

Dopo che la funzione è stata definita e il suo contenuto assegnato ad una variabile posso usare la variabile esattamente come se avessi definito una funzione con nome.

Prendiamo l'esempio precedente. Possiamo ottenere il medesimo risultato utilizzando una funzione anonima:

```
stop();
//semplice timer che misura il tempo trascorso
//dal lancio del filmato - utilizza una funzione anonima

// assegno alla variabile my_timer una funzione
// e che aggiorna il contenuto del campo di testo
// messaggio_txt sulla base di quanto restituito dalla
// funzione incorporata getTimer

var my_timer:Function = function ():Void {
    //millisecondi trascorsi
    var t:Number = getTimer();

    //secondi = parte intera delle divisione per 1000
    var s:Number = Math.floor(t/1000);

    //millesimi = resto della divisione per 1000
    var m:Number = t % 1000;

    messaggio_txt.text = s + " secondi e " + m +
        " millesimi.";
}
// la funzione my_timer viene eseguita una prima volta
// appena il filmato viene eseguito

my_timer();

// utilizzazndo il timer del computer la funzione my_timer
// viene
// eseguita ogni 30 millisecondi
setInterval(my_timer, 30);
```

L'esempio qui riportato si trova nel file FunzioneAnonima\_1 fla.

---

Notate la differenza tra `function` (con la iniziale minuscola) che è l'istruzione che consente di definire una funzione e `Function` (con la iniziale maiuscola) che indica il tipo `Function` e comunica al compilatore che la variabile `my_timer` conterrà una funzione. In `ActionScript` non è obbligatorio, quando si dichiara una variabile (con l'istruzione `var`), dichiarare il tipo di dati che la variabile è destinata a contenere, ma è fortemente consigliato.

Nel caso illustrato posso usare indifferentemente una funzione anonima o una funzione con nome: il risultato pratico è praticamente identico. Normalmente viene scelta la prima versione (funzione con nome) perché più leggibile.

In realtà c'è un po' di differenza tra l'uso di una funzione con nome e di una variabile che contiene una funzione anonima. Se per esempio modifico il codice in cui uso la funzione con nome mettendo la definizione della funzione infondo così:

```
stop();
//semplice timer che misura il tempo trascorso dal lancio del filmato

// la funzione my_timer viene eseguita una prima volta
// appena il filmato viene eseguito

my_timer(); // funziona!!!!

// poi utilizzando il timer del computer la funzione
// my_timer viene eseguita ogni 30 millisecondi

setInterval(my_timer, 30);

//definisco la funzione my_timer() che non ritorna alcun valore e che aggiorna il
//contenuto del campo di testo messaggio_txt sulla base di quanto restituito dalla
//funzione incorporata getTimer

function my_timer ():Void {
    //millisecondi trascorsi
    var t:Number = getTimer();
    //secondi = parte intera delle divisione
    var s:Number = Math.floor(t/1000);
    //millesimi = resto della divisione
    var m:Number = t % 1000;
    messaggio_txt.text = s + " secondi e " + m + " millesimi.";
}
```

Il mio codice funzionerà lo stesso. Mentre se apporto una modifica simile al codice in cui ho usato la funzione anonima, così:

```
stop();
//semplice timer che misura il tempo trascorso dal lancio del filmato

//la funzione my_timer viene eseguita una prima volta appena il filmato viene eseguito

my_timer(); //errore!!!!

// utilizzando il timer del computer la funzione my_timer viene
// eseguita ogni 30 millisecondi
setInterval(my_timer, 30);

//assegno alla variabile my_timer una funzione che aggiorna il contenuto del campo di
//testo messaggio_txt sulla base di quanto restituito dalla funzione getTimer

var my_timer:Function = function ():Void {
    //millisecondi trascorsi
    var t:Number = getTimer();

    //secondi = parte intera delle divisione per 1000
    var s:Number = Math.floor(t/1000);

    //millesimi = resto della divisione per 1000
    var m:Number = t % 1000;
    messaggio_txt.text = s + " secondi e " + m +
        " millesimi.";
}
```



Il compilatore mi riporterà un errore. Nel primo caso `my_timer` è una funzione e viene compilata (come se fosse un modulo separato) prima dell'altro codice. Nel secondo caso `my_timer` è una variabile e non contiene nulla fino a che non gli è stato assegnato un valore.

Ci sono casi in cui l'uso delle funzioni anonime è necessario. Il caso più tipico è quando devo gestire un evento legato ad un oggetto. In questo la sintassi è la seguente:

```
oggetto.evento = function (parametro1, parametro2, ...) {
    // Blocco di istruzioni che gestiscono l'evento
}
```

Questa porzione di codice comprende le parti seguenti:


- `oggetto` è il nome di oggetto il cui evento deve essere gestito.
- Evento è un evento che l'oggetto *spara* in determinate condizioni (nel caso di un bottone può essere, ad esempio, `onRelease`, `onPress`, `onRollOver`, ecc.)
- `parametro1`, `parametro2`, ... uno o più parametri che vengono passati alla funzione gestore d'evento. I parametri sono detti anche *argomenti*.
- Blocco di istruzioni contiene tutto il codice ActionScript che deve essere eseguito quando si verifica l'evento.

Nell'esempio seguente scriveremo il codice che carica un suono, lo esegue quando il caricamento è terminato e segnala con appositi messaggi sullo schermo quando il suono inizia e finisce.

Come prima cosa dovremo creare sullo stage un campo di testo. Per fare questo utilizzeremo l'apposito tool presente nella *palette strumenti* e utilizzando la *palette proprietà* definiremo le caratteristiche del campo.

Il campo sarà un campo di tipo *testo dinamico* e servirà a mostrare sullo schermo l'output del programma ossia i messaggi che segnaleranno che il suono è iniziato e terminato: come font del campo sceglieremo `_sans` e gli daremo il nome `messaggio_txt`.

A questo punto aggiungeremo un livello nella linea temporale che chiameremo *Azioni*.

Selezioniamo il primo frame sul livello azioni e apriamo la finestra delle azioni cliccando sulla palette delle proprietà il simbolo . Sulla barra della finestra delle azioni dovrà essere scritto: *Azioni - Fotogramma* ad indicare che stiamo modificando le azioni per quel fotogramma della linea temporale.

Inseriamo il codice seguente:

```

stop();
//uso di una funzione anonima per gestire eventi

//creo un'istanza della classe sound

var mio_suono:Sound = new Sound();
//la variabile mio_suono contiene ora un'istanza
//della classe sound

//definisco la funzione che viene eseguita
//quando il suono viene caricato completamente
mio_suono.onLoad = function(success) {
    if (success) {
        //aggiorno il testo
        messaggio_txt.text = "Il suono è in esecuzione."
        //faccio partire il suono
        this.start();
    } else {
        //se success è falso significa che c'è
        //stato un errore
        messaggio_txt.text = "Si è verificato un errore!"
    }
}

//definisco la funzione che viene eseguita
//quando l'esecuzione del suono è completata
mio_suono.onSoundComplete = function() {
    //aggiorno il testo
    messaggio_txt.text = "Esecuzione completata.";
}

//carico il suono nell'oggetto mio_suono
mio_suono.loadSound("Round.mp3", false);

```

L'esempio qui riportato si trova nel file FunzioneAnonima\_2 fla.

Definizione di funzioni globali e associate alla linea temporale  
 Le funzioni sono associate alla linea temporale del clip filmato che le definisce. È quindi necessario utilizzare un percorso target per chiamarle.

È possibile utilizzare l'identificatore `_global` per dichiarare una funzione globale che sia disponibile per tutte le linee temporali e le aree di validità senza l'uso di un percorso target.

Per definire una funzione globale, è necessario definire una funzione anonima e assegnarla a una variabile preceduta dal percorso `_global` come nell'esempio seguente:

```

_global.myFunction = function(myNum:Number):Number {
    return (myNum * 2) + 3;
};
trace(myFunction(5)) // 13

```

Dopo che è stata definita la funzione `myFunction` potrà essere richiamata da qualsiasi riga di codice `ActionScript` senza specificare alcun percorso target come succede per le funzioni incorporate.

Per definire una funzione della linea temporale, utilizzare l'istruzione `function` seguita dal nome della funzione, dai parametri da passare alla funzione e dalle istruzioni `ActionScript` che indicano le operazioni da essa eseguite.

Nell'esempio seguente viene definita una funzione denominata `areaOfCircle` con il parametro `radius`:

```
function areaOfCircle(radius:Number):Number {
    return (Math.PI * radius * radius);
}
trace (areaOfCircle(8));
```

La funzione sarà disponibile a partire dal frame in cui è stata definita e potrà essere richiamata senza specificare un percorso target dal `MovieClip` corrente. Per richiamare la funzione da un altro `MovieClip` sarà necessario specificare un percorso target relativo o assoluto.

Se, ad esempio, la funzione `areaOfCircle` è stata definita nel `MovieClip` `mycircle_mc` che è collocato nel filmato principale, potrà essere raggiunta (percorso target assoluto) scrivendo:

```
var r:Number = 7;
var Area:Number = root.mycircle_mc.areaOfCircle(r);
```

#### Uso di variabili nelle funzioni

Le variabili locali sono strumenti utili per organizzare il codice e renderlo più comprensibile. Le variabili locali hanno come area di validità il corpo della funzione e cessano di esistere al termine della stessa. Qualsiasi parametro passato a una funzione viene considerato una variabile locale.

Facciamo un esempio:

1. Creare un nuovo documento Flash e salvarlo come `flashvariables fla`.
2. Aggiungere il codice `ActionScript` seguente al fotogramma 1 della linea temporale principale:

```
var myName:String = "Ester";
var myAge:String = "65";
var myFavSoftware:String = "Flash";
function traceMe(yourFavSoftware:String, yourName:String,
yourAge:String) {
    trace("Sono " + yourName + ", apprezzo " + yourFavSoftware +
        " e ho " + yourAge + " anni.");
}
```

```
}  
traceMe(myFavSoftware, myName, myAge);
```

3. Selezionare Controllo > Prova filmato per provare il documento Flash.

Come possiamo vedere i parametri, detti anche *argomenti*, sono gli elementi che il codice della funzione elabora per produrre il suo risultato.

Passaggio di parametri a una funzione

Si possono passare più parametri ad una funzione separandoli con delle virgole.

Talvolta i parametri sono obbligatori e talvolta sono facoltativi. In una funzione potrebbero essere presenti sia parametri obbligatori che opzionali.

In ogni caso se si passa alla funzione un numero di parametri inferiore a quelli dichiarati, Flash imposta i valori dei parametri mancanti a undefined. Questo può provocare risultati imprevisti:

1. Creare un nuovo documento Flash e salvarlo come Somma\_0 fla.
2. Aggiungere il codice seguente al fotogramma 1 della linea temporale principale:

```
function somma(a:Number, b:Number, c:Number):Number {  
    return (a + b + c);  
}  
// sommo tre numeri  
trace(somma(1, 4, 6)); // 11  
// La somma non è un numero (c è uguale a undefined)  
trace(somma(1, 4)); // NaN  
// il parametro non dichiarato è ignorato  
trace(addNumbers(1, 4, 6, 8)); // 11
```

3. Selezionare Controllo > Prova filmato per provare il documento Flash. Flash visualizza i seguenti valori nella finestra di output: 11, NaN, 11.

Se non si passa un numero sufficiente di parametri alla funzione `somma`, agli argomenti mancanti verrà assegnato il valore predefinito `undefined`. Se si passano troppi parametri, quelli in eccesso verranno ignorati.

È possibile scrivere funzioni con numero di parametri variabile. Prendiamo l'esempio in Somma fla:

```
//definisco una funzione somma che somma fra loro  
//un numero qualsiasi di numeri interi  
  
function somma():Number {  
    var sum:Number = 0;  
    // arguments è una speciale variabile che  
    // contiene in un array i parametri passati  
    // alla funzione  
    // somma a sum tutti i numeri passati come parametri  
    for (var i = 0; i < arguments.length; i++) {  
        // controllo che il parametro passato sia un  
        // numero se no lo ignoro
```

```

        if (! isNaN(arguments[i])) {
            sum = sum + arguments[i];
        }
    }
    // restituisco il valore di sum
    return (sum);
}

trace (somma()); // 0
trace (somma(5,6,7,20)); //38
trace(somma(10,7)); //17

trace (somma(10,"pippo",89)) //99;

```

Per scrivere funzioni che possano avere un numero qualsiasi di parametri devo usare nel corpo della funzione al posto dei parametri dichiarati la variabile speciale `arguments` che organizza in un Array i parametri passati alla funzione.

In questi casi i parametri non vengono dichiarati e quindi il controllo del tipo (il controlla, cioè che vengano passati parametri coerenti alle azioni che compie la funzione) sono a carica del programmatore.

### Restituzione di valori da funzioni

Una funzione può restituire un valore che di norma è il risultato dell'operazione compiuta. Per compiere questa operazione si utilizza l'istruzione `return` che specifica il valore che verrà restituito dalla funzione.

L'istruzione `return` ha anche l'effetto di interrompere immediatamente il codice in esecuzione nel corpo della funzione e restituire immediatamente il controllo del flusso di programma al codice chiamante.

Nell'utilizzo dell'istruzione `return` si applicano le regole seguenti:

- Se per una funzione si specifica un tipo restituito diverso da `Void`, è necessario includere un'istruzione `return` seguita dal valore restituito dalla funzione.
- Se si specifica un tipo restituito `Void`, non occorre includere un'istruzione `return`. Se l'istruzione `return` viene specificata, non deve essere seguita da valori.
- Indipendentemente dal tipo restituito, un'istruzione `return` può essere utilizzata per uscire da una funzione e restituire il controllo al codice chiamante
- Se non si specifica un tipo `return`, l'inclusione di un'istruzione `return` è opzionale.

La funzione seguente restituisce ad esempio il quadrato del parametro `myNum` e specifica che il valore restituito deve essere di tipo `Number`:

```

function sqr(myNum:Number):Number {
    return myNum * myNum;
}

```

Alcune funzioni eseguono una serie di operazioni senza restituire un valore.

Il seguente esempio restituisce il valore elaborato. Il valore viene memorizzato in una variabile che può essere utilizzata all'interno dell'applicazione:

1. Creare un nuovo documento Flash e salvarlo come return fla.
2. Aggiungere il codice seguente al fotogramma 1 della linea temporale principale:

```
//La funzione getArea accetta due parametri: width e height.  
//entrambi numeri  
  
function getArea(width:Number, height:Number):Number {  
    return width * height;  
}
```

3. Immettere il codice seguente dopo la funzione:

```
var area:Number = getArea(10, 12);  
trace(area); // 120
```

La chiamata alla funzione getArea() assegna i valori 10 e 12 rispettivamente ai parametri width e height e il valore restituito viene salvato nella variabile area. Il valore salvato nella variabile area vengono quindi scritti sulla finestra di output con il comando trace.

4. Selezionare Controllo > Prova filmato per provare il file SWF.

Nel pannello Output viene visualizzato 120.

I parametri nella funzione getArea() sono analoghi ai valori di una variabile locale, ovvero esistono fintanto che la funzione viene chiamata e cessano di esistere quando la funzione termina.

Nozioni fondamentali sui metodi

I metodi sono funzioni associate a una classe che può essere una classe scritta da voi o incorporata, ovvero parte del linguaggio ActionScript. Le funzioni incorporate in una classe si chiamano metodi perché sono strumenti che ci fornisce l'oggetto per svolgere il suo compito.

Vediamo ora un esempio di come si utilizza un metodo di una classe incorporata. Per farlo useremo alcuni metodi associati alla classe Array:

1. Creo un nuovo documento Flash e salvarlo come metodi fla.
2. Aggiunge il codice seguente nel primo fotogramma della linea temporale:

```
var userArr:Array = new Array();  
userArr.push({firstname:"Giorgio", age:39});
```

---

```
userArr.push({firstname:"Daniele", age:43});
userArr.push({firstname:"Luca", age:2});
userArr.sortOn("firstname");
var userArrayLenth:Number = userArr.length;
var i:Number;
for (i = 0; i < userArrayLenth; i++) {
    trace(userArr[i].firstname);
}
```

Viene creato un nuovo oggetto Array denominato userArr. Viene utilizzato il metodo push() della classe Array per aggiungere all'array tre oggetti che, a loro volta, contengono nome ed età. Utilizzando il metodo sortOn() l'array viene ordinato in base al valore della proprietà firstname di ogni oggetto. Utilizzando un ciclo di iterazione gestito con for, si visualizza il nome di ogni elemento dell'array nel pannello Output . I nomi vengono visualizzati in ordine alfabetico.

3. Selezionare Controllo > Prova filmato per provare il file SWF.





## Le classi

I linguaggi orientati agli oggetti si basano sul concetto di *classi* e *interfacce*. Una classe definisce tutte le proprietà e i comportamenti che distinguono una serie di oggetti.

Se vogliamo organizzare i diversi utenti che usano un'applicazione possiamo creare una classe *User*, ad esempio, che rappresenta gli utenti. Nella definizione della classe vengono definite le proprietà che hanno gli utenti (ad esempio nome, cognome, username, password, ecc.). Per gestire i singoli utenti vengono create istanze della classe, che, per la classe *User*, rappresentano ogni singolo individuo, ovvero, come si dice, i membri della classe.

Le istanze della classe *User* comprendono tutte le proprietà della classe *User*.

Quando abbiamo parlato dei tipi di dati abbiamo visto che esistono tipi di dati primitivi e tipi di dati derivati o complessi. In linea di principio in un linguaggio orientato agli oggetti tutti i tipi di dati sono classi. Creando una nuova classe il programmatore crea un nuovo tipo di dati complesso, un modello, cioè, delle proprietà e delle caratteristiche che ha un nuovo tipo di oggetto. Per creare un tipo di dati *Lattuga* dovrò scrivere la classe *Lattuga* all'interno della quale definirò ad esempio le proprietà *tipo\_foglia* e *colore* e il metodo *lavare()*. In questo modo si definisce l'oggetto *Lattuga* alle cui istanze potranno essere applicato il metodo *lavare()* e potranno essere impostate le proprietà *tipo\_foglia* e *colore*.

## Nozioni di base

### Oggetti

Anche un oggetto del mondo reale, ad esempio un gatto, possiede delle *proprietà* (o stati), quali nome, età e colore, e presenta delle caratteristiche comportamentali, ad esempio dormire, mangiare e fare le fusa.

Analogamente, nel mondo della programmazione orientata agli oggetti, gli oggetti presentano proprietà e comportamenti. Utilizzando le tecniche orientate agli oggetti, è possibile modellare un oggetto del mondo reale (come un gatto) oppure un oggetto più astratto (ad esempio un processo chimico).

---

## Istanze e membri di classe

Procedendo con l'analogia con un gatto nel mondo reale, è possibile considerare che esistono gatti di colore, età e nomi differenti che allo stesso tempo presentano modi diversi di mangiare o di fare le fusa. Tuttavia, indipendentemente dalle differenze tra i singoli animali, tutti i gatti appartengono alla stessa categoria che, in termini di programmazione orientata agli oggetti, è detta classe. Appartengono cioè alla classe Gatto. Nella terminologia orientata agli oggetti, ogni singolo gatto viene considerato un'*istanza* della classe Gatto.

Nella programmazione orientata agli oggetti una classe definisce un modello per un tipo di oggetto. Tutte le caratteristiche e i comportamenti che appartengono a una classe sono detti membri di tale classe. In particolare, le caratteristiche (nell'esempio del gatto, il nome, l'età e il colore) sono dette proprietà della classe e sono rappresentate da variabili, mentre i comportamenti (mangiare, dormire) sono detti metodi della classe e sono rappresentati da funzioni.

## Ereditarietà

Uno dei vantaggi principali della programmazione orientata agli oggetti è rappresentato dalla creazione di sottoclassi (tramite estensione di una classe); una sottoclasse eredita tutte le proprietà e i metodi della rispettiva classe. La sottoclasse definisce generalmente ulteriori metodi e proprietà o sostituisce metodi o proprietà definiti nella superclasse. Le sottoclassi possono inoltre sostituire i metodi definiti in una superclasse, ovvero fornire definizioni proprie per tali metodi.

Uno dei vantaggi principali derivanti dall'uso di una struttura superclasse/sottoclasse è la possibilità di riutilizzare in modo semplice il codice simile tra diverse classi. È possibile, ad esempio, creare una superclasse denominata Animale che contiene caratteristiche e comportamenti comuni di tutti gli animali, quindi creare diverse sottoclassi che ereditano dalla superclasse Animale e aggiungono caratteristiche e comportamenti specifici di un determinato tipo di animale.

La classe Gatto di cui abbiamo parlato potrebbe essere l'estensione di una superclasse dalla quale eredita da proprietà e metodi. È possibile, ad esempio, creare una classe Mammifero che definisce alcune proprietà e alcuni comportamenti comuni a tutti i mammiferi e, successivamente, creare una sottoclasse Gatto che estenda la classe Mammifero.

Un'ulteriore sottoclasse, ad esempio la classe Siamese, potrebbe a sua volta estendere (creando una *sottoclasse*) la classe Gatto, e così via.

La creazione di sottoclassi consente di riutilizzare il codice. Invece di ricreare tutti i codici comuni a entrambe le classi, è possibile estendere semplicemente la classe esistente.

---

## Interfacce

Le *interfacce* nella programmazione orientata agli oggetti possono essere descritte come modelli di definizioni di classe; le classi che implementano le interfacce sono necessarie per implementare quel modello di metodo.

Usando l'analogia del gatto, un'interfaccia è simile al progetto di un gatto: Il progetto evidenzia quali parti sono necessarie, ma non necessariamente come vengono assemblate le parti o qual è il funzionamento delle parti.

È possibile utilizzare interfacce per migliorare la struttura e facilitare la manutenzione delle applicazioni. Poiché ActionScript 2.0 supporta l'estensione da una sola superclasse, le interfacce possono essere utilizzate come forma limitata di ereditarietà multipla.

Un'interfaccia può anche essere considerata un "contratto di programmazione", utilizzabile per creare relazioni tra classi altrimenti non correlate. Ad esempio, si consideri una situazione di lavoro con un team di programmatori, ciascuno dei quali si occupa di una sezione (classe) diversa della stessa applicazione. In fase di progettazione dell'applicazione, viene stabilito un set di metodi che le classi utilizzeranno per comunicare. Viene quindi creata un'interfaccia che dichiara questi metodi, i relativi parametri e tipi restituiti. Qualsiasi classe che implementa questa interfaccia deve fornire le definizioni per tali metodi; in caso contrario si verificherà un errore

## Incapsulamento

Nella progettazione orientata agli oggetti, gli oggetti sono considerati scatole nere che contengono o *incapsulano* funzionalità. Un programmatore dovrebbe essere in grado di interagire con un oggetto conoscendone solo le proprietà, i metodi e gli eventi (l'interfaccia di programmazione), ma senza conoscerne i dettagli di implementazione. Questo approccio consente di programmare a un livello di astrazione superiore e garantisce una struttura organizzativa per la creazione di sistemi complessi.

Per garantire l'incapsulamento, ActionScript 2.0 comprende, ad esempio, il controllo dell'accesso dei membri, affinché i dettagli dell'implementazione possano essere resi privati e invisibili al codice esterno a un oggetto che deve interagire con l'interfaccia di programmazione dell'oggetto, anziché con i relativi dati di implementazione (che possono essere nascosti nei metodi e nelle proprietà privati). Questo approccio garantisce alcuni vantaggi importanti: consente ad esempio al creatore dell'oggetto di modificare l'implementazione dell'oggetto senza dover apportare modifiche al codice esterno all'oggetto, a condizione che l'interfaccia di programmazione non cambi.

## Polimorfismo

La programmazione orientata agli oggetti permette di esprimere differenze tra le singole classi con l'ausilio di una tecnica detta polimorfismo, grazie

---

alla quale le classi possono sostituire i metodi delle relative superclassi e definire implementazioni specializzate di tali metodi.

È possibile, ad esempio, creare una classe Mammifero che disponga dei metodi giocare() e dormire(), quindi creare sottoclassi Gatto, Scimmia e Cane che estendano la classe Mammifero. Le sottoclassi sostituiscono il metodo giocare() della classe Mammifero per riflettere le abitudini di ogni animale. Scimmia implementa il metodo giocare() per passare da un albero a un altro, Gatto implementa il metodo giocare() per far rimbalzare un gomitolo di lana, mentre Cane implementa il metodo giocare() per rincorrere un palla.

Dato che la funzionalità dormire() è simile per tutti gli animali, si utilizza l'implementazione della superclasse.

---

## Object, Array e Date

Iniziamo il nostro percorso tra le classi incorporate nel linguaggio ActionScript da tre classi di base presenti, con piccole differenze in tutti i linguaggi orientati agli oggetti. Per informazione sulle altre classi incorporate rimandiamo a "Guida di riferimento di ActionScript 2.0", presente sul CD allegato alla dispensa.

### Object

Come abbiamo visto un oggetto è un insieme di proprietà. Una *proprietà* è un attributo che descrive lo stato dell'oggetto. La trasparenza di un oggetto, quale un clip filmato, ad esempio, è un attributo che descrive l'aspetto dell'oggetto. *alpha* (trasparenza) è quindi una proprietà. Ogni proprietà è provvista di un nome e di un valore. Il valore di una proprietà può essere qualsiasi tipo di dati di Flash, anche il tipo di dati *Object*. Di conseguenza è possibile disporre oggetti all'interno di altri oggetti, ovvero *nidificarli*.

Per specificare gli oggetti e le relative proprietà, usare l'operatore punto (.). Nel codice seguente, ad esempio, *hoursWorked* è una proprietà di *weeklyStats*, che è a sua volta una proprietà di *employee*:

```
employee.weeklyStats.hoursWorked
```

L'oggetto *MovieClip* di ActionScript dispone di metodi che consentono di controllare le istanze del simbolo clip filmato nello stage. In questo esempio sono utilizzati i metodi *play()* e *nextFrame()*:

```
mcInstanceName.play();  
mc2InstanceName.nextFrame();
```

Normalmente è la classe che definisce metodi e proprietà degli oggetti membri.

Il tipo di dati *Object* rappresenta l'astrazione dell'idea di oggetto. Sta alla radice della gerarchia delle classi: tutte le classi sono estensioni di *Object*.

Oltre a definire le proprietà e i metodi base di ogni classe, la classe *Object* mi consente di creare oggetti vuoti, senza proprietà né metodi. Creando un'istanza di *Object* creo un oggetto personalizzato, a cui posso aggiungere

---

le proprietà che voglio e che mi può servire per organizzare le informazioni nell'applicazione Flash.

In una applicazione di shopping on-line potrei avere bisogno di diverse informazioni: ad esempio, potrebbero essere necessari il nome, l'età e il numero di telefono dell'utente, la velocità di un pallone, il nome degli articoli contenuti in un carrello, il numero di fotogrammi caricati o l'ultimo tasto premuto. La creazione di oggetti personalizzati consente di organizzare le informazioni in gruppi e di semplificare e riutilizzare gli script.

Nel codice ActionScript seguente è contenuto un esempio dell'uso di oggetti personalizzati per l'organizzazione di informazioni. Vengono creati un nuovo oggetto denominato `user` e tre proprietà: `name`, `age` e `phone` che sono tipi di dati `String` e `Numeric`.

```
var user:Object = new Object();
user.name = "Irving";
user.age = 32;
user.phone = "555-1234";
```

Lo stesso oggetto può essere creato anche assegnando alla variabile il letterale di tipo `Object` corrispondente.

```
var user:Object;
user = {name:"Irving",age:32,phone:"555-1234"};
```

In questa sintassi non è necessario richiamare il costruttore della classe `Object()` con l'operatore `new`. In questo corso, comunque, privilegeremo sempre la prima sintassi perché più chiara e coerente con l'utilizzo delle altre classi.

Un altro esempio interessante dell'uso del tipo `Object` lo trovi in `metodi fla`:

```
var userArr:Array = new Array();
userArr.push({firstname:"Giorgio", age:39});
userArr.push({firstname:"Daniele", age:43});
userArr.push({firstname:"Luca", age:2});
userArr.sortOn("firstname");
var userArrayLenth:Number = userArr.length;
var i:Number;
for (i = 0; i < userArrayLenth; i++) {
    trace(userArr[i].firstname);
}
```

Qui viene usata la sintassi abbreviata di creazione di un oggetto per assegnare gli oggetti ad elementi di un `Array`.

---

## Array

Un *array* è un oggetto le cui proprietà sono identificate da un numero che ne rappresenta la posizione nella struttura dell'array. Un array è fondamentalmente un elenco di elementi. È importante tenere a mente che non occorre che gli elementi dell'array abbiano lo stesso tipo di dati. È possibile inserire numeri, dati, stringhe, oggetti e anche aggiungere un array nidificato per ogni indice di array.

L'esempio seguente mostra un array semplice di nomi di mesi.

```
var myArr:Array = new Array();  
myArr[0] = "January";  
myArr[1] = "February";  
myArr[2] = "March";  
myArr[3] = "April";
```

L'array precedente può essere riscritto come segue:

```
var myArr:Array = new Array("January", "February", "March",  
    "April");
```

In alternativa, come per il tipo Object, è possibile assegnare direttamente alla variabile un letterale di tipo Array:

```
var myArr:Array = ["January", "February", "March", "April"];
```

Un array può essere paragonato a un edificio ad uso uffici, dove ogni piano contiene dati diversi, ad esempio la contabilità al terzo piano e la progettazione al quinto piano. È possibile memorizzare diversi tipi di dati in un solo array, compresi altri array. Ogni piano dell'edificio può contenere più tipi di contenuto, ad esempio la direzione (*executives*) e la contabilità (*accounting*) possono trovarsi entrambi al terzo piano.

Un array contiene *elementi* che equivalgono a ogni piano dell'edificio. Ogni elemento ha una posizione numerica (l'*indice*) che corrisponde alla posizione di ogni elemento dell'array come ogni piano dell'edificio ha un numero.

Ogni elemento può contenere dati (vale a dire un numero, una stringa, un valore booleano, un array, un oggetto) oppure essere vuoto.

L'array può inoltre essere controllato e modificato. Potrebbe ad esempio essere necessario spostare il reparto progettazione al piano terra dell'edificio, aggiungere od eliminare un piano. Analogamente si possono spostare i valori in punti diversi dell'array e modificarne le dimensioni.

L'edificio (l'array) contiene piani (gli elementi), ogni piano è numerato (l'indice) e contiene uno o più reparti (i valori).

---

Potete trovare il file di esempio, array.fla, nella cartella Esempi. L'esempio illustra la gestione di array con ActionScript. Il codice dell'esempio crea un array; quindi ordina, aggiunge e rimuove le voci di due componenti List.

## Uso degli array

Gli array possono essere utilizzati in modi diversi. Se si caricano dati dal un database collocato su un server Web remoto, probabilmente ci converrà organizzare i dati sotto forma di array di oggetti.

Immaginiamo, ad esempio, di costruire un jukebox: la sequenza dei brani potrebbe essere memorizzata come un array di informazioni sui brani, a loro volta organizzate in oggetti. Ogni oggetto potrebbe contenere il nome del brano, il nome dell'artista, la durata, il percorso del file audio (ad esempio MP3) o altre informazioni che si desidera associare a un determinato brano.

La posizione di un elemento nell'array è detta *indice*. Tutti gli array sono con base zero, ovvero [0] è il primo elemento dell'array, [1] è il secondo, e così via.

Esistono diversi tipi di array, come illustrato nelle sezioni seguenti. Gli array più comuni utilizzano un indice numerico per la ricerca di un determinato elemento in un *array indicizzato*. Il secondo tipo di array è detto *array associativo* e utilizza un indice di testo anziché un indice numerico per la ricerca di informazioni.

La classe incorporata Array consente di accedere agli array e di manipolarli. Per creare un oggetto Array, utilizzare la funzione di costruzione new con il costruttore Array() o l'operatore di accesso agli array ([]). Nell'esempio seguente viene costruito un array indicizzato.

1. Creare un nuovo documento Flash e salvarlo come basicArrays.fla.
2. Aggiungere il codice ActionScript seguente al fotogramma 1 della linea temporale:

```
// Definisce un nuovo array
var myArr:Array = new Array();
// Definisce valori su due indici
myArr[1] = "value1";
myArr[0] = "value0";
// Esegue un'iterazione sugli elementi dell'array
var i:String;
for (i in myArr) {
// Traccia le coppie chiave/valore
trace("key: " + i + ", value: " + myArr[i]);
}
```

Nella prima riga del codice ActionScript viene definito un nuovo array per contenere i valori, quindi si definiscono dati (value0 e value1) su due indici

---



dell'array. Si utilizza un ciclo for..in per eseguire iterazioni su ciascun elemento dell'array e visualizzare le coppie chiave/valore nel pannello Output utilizzando un'istruzione trace.

3. Selezionare Controllo > Prova filmato per provare il codice.

Nota bene che il medesimo array poteva essere creato utilizzando l'operatore di accesso agli array ([]). In questo caso il codice sarebbe risultato così:

```
// Definisce un nuovo array
var myArr:Array = ["value1","value0"];
// Definisce valori su due indici
// Esegue un'iterazione sugli elementi dell'array
var i:String;
for (i in myArr) {
// Traccia le coppie chiave/valore
trace("key: " + i + ", value: " + myArr[i]);
}
```

Il costrutto è più sintetico ma meno leggibile e didatticamente meno coerente alla sintassi di creazione delle istanze di una classe, per cui in questo corso utilizzeremo sempre la prima versione.

### Modifica di un array

L'array può essere controllato e modificato tramite ActionScript. È possibile spostare valori all'interno dell'array o modificarne la dimensione. Il seguente codice, ad esempio, scambia i dati di due indici di un array:

```
var buildingArr:Array = new Array();
buildingArr[2] = "Accounting";
buildingArr[4] = "Engineering";
trace(buildingArr);
//undefined,undefined,Accounting,undefined,Engineering
var temp_item:String = buildingArr[2];
buildingArr[2] = buildingArr[4];
buildingArr[4] = temp_item;
trace(buildingArr);
//undefined,undefined,Engineering,undefined,Accounting
```

Nell'esempio precedente è necessario creare una variabile temporanea perché se il contenuto dell'indice 4 dell'array fosse stato copiato nell'indice 2 dell'array senza salvarne prima il contenuto, il contenuto originale dell'indice 2 sarebbe andato perso.

Con il codice seguente, ad esempio, il valore dell'indice 2 dell'array (Accounting) andrebbe perso:

```
// Modalità scorretta; non viene utilizzata la variabile
temporanea
buildingArr[2] = buildingArr[4];
buildingArr[4] = buildingArr[2];
```

```
trace(buildingArr);  
//undefined,undefined,Engineering,undefined,Engineering
```

## Lunghezza di un array

Quando si lavora con gli array, è spesso necessario conoscere il numero degli elementi contenuti in un array, in particolare se si scrivono cicli for che eseguono iterazioni su ogni elemento dell'array ed eseguono una serie di istruzioni. La proprietà `length` restituisce la lunghezza dell'array.

Qui di seguito troverai due esempi:

Nel primo esempio, viene creato un array che contiene i nomi dei mesi. Vengono visualizzati il contenuto e la lunghezza dell'array. Un ciclo for esegue un'iterazione su ogni elemento dell'array e converte il valore in maiuscole. Il contenuto dell'array viene quindi nuovamente visualizzato.

```
var monthArr:Array = new Array("Jan", "Feb", "Mar", "Apr",  
    "May", "Jun",  
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");  
trace(monthArr); //  
    Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec  
trace(monthArr.length); // 12  
var i:Number;  
for (i = 0; i < monthArr.length; i++) {  
    monthArr[i] = monthArr[i].toUpperCase();  
}  
trace(monthArr);  
    //JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
```

Nel secondo esempio se si crea un elemento all'indice 5 dell'array, la lunghezza dell'array restituisce 6 (l'array ha base zero) e non il numero effettivo di elementi contenuti nell'array: questo perché gli elementi dall'indice 0 all'indice 4 valgono `undefined`. Se, cioè, assegnando valori ad elementi di un array salto degli indici, gli elementi saltati esisteranno comunque e varranno `undefined`.

```
var myArr:Array = new Array();  
myArr[5] = "five";  
trace(myArr.length); // 6  
trace(myArr);  
    //undefined,undefined,undefined,undefined,undefined,5
```

## Aggiunta e rimozione di elementi

Un array contiene elementi e ogni elemento ha una posizione numerica (l'indice) che corrisponde alla modalità con cui si fa riferimento alla posizione di ogni elemento nell'array.

Ogni elemento può contenere dati o essere vuoto. I dati contenuti possono essere del formato numerico, stringa, booleano o essere un array o un oggetto.

---

Quando si creano elementi in un array, si consiglia di creare gli indici in modo sequenziale. Abbiamo appena visto che se si assegna un solo valore in un array all'indice 5, la lunghezza dell'array restituisce 6. Nell'array vengono quindi inseriti cinque valori non definiti.

L'esempio seguente dimostra come creare un nuovo array, eliminare un elemento da un determinato indice e aggiungere e sostituire dati in corrispondenza di un indice di un array:

```
var monthArr:Array = new Array("Jan", "Feb", "Mar", "Apr",
    "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
delete monthArr[5];
trace(monthArr);
    //Jan,Feb,Mar,Apr,May,undefined,Jul,Aug,Sep,Oct,Nov,Dec
trace(monthArr.length); // 12
monthArr[5] = "JUN";
trace(monthArr);
    //Jan,Feb,Mar,Apr,May,JUN,Jul,Aug,Sep,Oct,Nov,Dec
```

Anche se è stato eliminato l'elemento all'indice 5 dell'array, la lunghezza dell'array rimane 12 e all'elemento all'indice 5 dell'array viene assegnato lo speciale valore `undefined` anziché essere eliminato totalmente.

### Array indicizzati

Come abbiamo visto gli array indicizzati memorizzano una serie di uno o più valori. Gli elementi possono essere recuperati in base alla loro posizione nell'array (indice). Il primo indice è sempre il numero 0 e viene incrementato di un'unità a ogni elemento successivo aggiunto all'array.

Il metodo `push()` della classe array consente di aggiungere elementi ad un array indicizzato.

Vediamo l'esempio seguente:

1. Creare un nuovo documento Flash e salvarlo come `indexArray fla`.
2. Aggiungere il codice ActionScript seguente al fotogramma 1 della linea temporale:

```
var myArray:Array = new Array();
myArray.push("one");
myArray.push("two");
myArray.push("three");
trace(myArray); // one,two,three
```

Nella prima riga del codice ActionScript viene definito un nuovo array per contenere i valori. Viene poi chiamato il metodo `push()` per aggiungere elementi all'array.

3. Selezionare **Controllo > Prova filmato** per provare il codice.

4. Tornare all'ambiente di creazione ed eliminare il codice nel pannello Azioni.

5. Al posto del codice eliminato inserire il codice ActionScript seguente:

```
var myArray:Array = ["one", "two", "three"];  
trace(myArray); // one,two,three
```

In questo codice si utilizza il valore letterale array (["one", "two", "three"]) per creare un nuovo array.

Questo codice è equivalente a quello scritto al punto 2. Quando si prova il codice, l'output visualizzato nel pannello Output è lo stesso.

### Creazione di array multidimensionali

In ActionScript è possibile implementare array *nidificati*, ovvero array di array. Questo tipo di array, detto anche *array multidimensionale* può essere paragonato a una matrice o una tabella e può pertanto essere utilizzato nella programmazione per rappresentare questi tipi di strutture.

Una scacchiera, ad esempio, è una griglia di otto colonne e righe e può essere rappresentata da un array contenente otto elementi, ognuno dei quali è a sua volta un array che contiene otto elementi.

Prendere in considerazione, ad esempio, un elenco di attività memorizzato come array indicizzato di stringhe:

```
var compiti:Array = ["lavare i piatti", "gettare  
l'immondizia"];
```

Se, ad esempio, si desidera memorizzare un elenco separato di attività per ogni giorno della settimana, è possibile creare un array multidimensionale con un elemento per ogni giorno della settimana.

Ogni elemento contiene un array indicizzato che a sua volta contiene l'elenco di attività:

1. Creare un nuovo documento Flash e salvarlo come multiArray1.fl.
2. Aggiungere il codice ActionScript seguente al fotogramma 1 della linea temporale:

```
var twoDArray:Array = new Array(new Array("one","two"), new  
    Array("three", "four"));  
trace(twoDArray);
```

Questo array, twoDArray, comprende due elementi che sono a loro volta array che contengono due elementi. In questo caso, twoDArray è l'array principale che contiene due array nidificati.

---

3. Selezionare Controllo > Prova filmato per provare il codice. I dati seguenti saranno visualizzati nel pannello Output:  
one,two,three,four
4. Tornare allo strumento di creazione e aprire il pannello Azioni. Impostare l'istruzione trace come commento (in questo modo il comando non verrà eseguito), come nell'esempio seguente:

```
// trace(twoDArray);
```

5. Aggiungere il seguente codice ActionScript dopo il codice del fotogramma 1 della linea temporale:

```
trace(twoDArray[0][0]); // uno
trace(twoDArray[1][1]); // quattro
```

Per recuperare elementi di un array multidimensionale, utilizzare gli operatori di accesso agli array ([]) dopo il nome dell'array di primo livello. Il primo [] fa riferimento all'indice dell'array di primo livello. Gli operatori di accesso agli array successivi fanno riferimento agli elementi degli array nidificati.

6. Selezionare Controllo > Prova filmato per provare il codice.

Per creare array multidimensionali è possibile utilizzare cicli for nidificati, come illustrato nell'esempio seguente.

1. Creare un nuovo documento Flash e salvarlo come multiArray2 fla.
2. Aggiungere il codice ActionScript seguente al fotogramma 1 della linea temporale:

```
var gridSize:Number = 3;
var mainArr:Array = new Array(gridSize);
var i:Number;
var j:Number;
for (i = 0; i < gridSize; i++) {
    mainArr[i] = new Array(gridSize);
    for (j = 0; j < gridSize; j++) {
        mainArr[i][j] = "riga " + i + " colonna " + j;
    }
}
trace(mainArr);
```

Questo codice ActionScript crea un array 3 x 3 e imposta il valore di ogni nodo dell'array. L'array (mainArr) viene quindi tracciato.

3. Selezionare Controllo > Prova filmato per provare il codice.

I dati seguenti saranno visualizzati nel pannello Output: riga 0 colonna 0, riga 0 colonna 1, riga 0 colonna 2, riga 1 colonna 0, riga 1 colonna 1, riga 1 colonna 2, riga 2 colonna 0, riga 2 colonna 1, riga 2 colonna 2

Per eseguire iterazioni su elementi di un array multidimensionale, si procede in maniera simile:

1. Creare un nuovo documento Flash e salvarlo come multiArray3.fla.
2. Copiare dall'esempio precedente il codice seguente e inserirlo al fotogramma 1 della linea temporale:

```
//codice copiato da multiArray2.fla
var gridSize:Number = 3;
var mainArr:Array = new Array(gridSize);
var i:Number;
var j:Number;
for (i = 0; i < gridSize; i++) {
    mainArr[i] = new Array(gridSize);
    for (j = 0; j < gridSize; j++) {
        mainArr[i][j] = "riga " + i + " colonna " + j;
    }
}
```

In questo codice, riportato nell'esempio precedente crea un array 3 x 3 e imposta il valore di ogni nodo dell'array

3. Aggiungere il seguente codice ActionScript al fotogramma 1 della linea temporale, di seguito al codice aggiunto al punto 2:

```
// Esegue un'iterazione sugli elementi
var outerArrayLength:Number = mainArr.length;
for (i = 0; i < outerArrayLength; i++) {
    var innerArrayLength:Number = mainArr[i].length;
    for (j = 0; j < innerArrayLength; j++) {
        trace(mainArr[i][j]);
    }
}
```

Questo codice ActionScript esegue iterazioni sugli elementi dell'array. Come condizione del ciclo viene utilizzata la proprietà length di ogni array. Il ciclo esterno esegue un'iterazione su ogni elemento di mainArray. Il ciclo interno esegue un'iterazione su ogni array nidificato e fornisce come output ogni nodo dell'array.

4. Selezionare Controllo > Prova. I dati seguenti verranno visualizzati nel pannello Output:

riga 0 colonna 0  
riga 0 colonna 1  
riga 0 colonna 2

---

riga 1 colonna 0  
riga 1 colonna 1  
riga 1 colonna 2  
riga 2 colonna 0  
riga 2 colonna 1  
riga 2 colonna 2

### Creazione di array associativi

Un array associativo è composto da *chiavi* e *valori* non ordinati e utilizza le chiavi alfanumeriche al posto degli indici numerici per organizzare i valori. Ogni chiave è una stringa univoca e viene utilizzata per accedere al valore a cui è associata. Il valore può essere un tipo di dati Number, Array, Object e così via.

In un'associazione tra una chiave e un valore si dice in genere che la chiave e il valore sono *mappati*. Una rubrica può essere considerata un array associativo dove le chiavi e i valori sono rappresentati, rispettivamente, dai nomi e dagli indirizzi e-mail.

Quando si utilizzano array associativi è possibile chiamare l'elemento dell'array desiderato utilizzando una stringa anziché un indice numerico. Dal punto di vista di ActionScript un array associativo è esattamente equivalente a un oggetto generico (tipo Object).

Questo mi consente di vedere, a seconda delle mie necessità un Object come array associativo e viceversa. Vediamo l'esempio seguente:

1. Creare un nuovo documento Flash e salvarlo come arrayAssociativo fla.
2. Immettere il codice ActionScript seguente nel fotogramma 1 della linea temporale:

```
// Definisce l'oggetto da utilizzare come array associativo
var someObj:Object = new Object();
// Definisce una serie di proprietà
someObj.myShape = "Rectangle";
someObj.myW = 480;
someObj.myH = 360;
someObj.myX = 100;
someObj.myY = 200;
someObj.myAlpha = 72;
someObj.myColor = 0xDFDFDF;
// Visualizza una proprietà utilizzando l'operatore punto e
// la sintassi di accesso agli array
trace(someObj.myAlpha); // 72
trace(someObj["myShape"]); // 72
```

La prima riga del codice ActionScript definisce un nuovo oggetto (someObj) che viene utilizzato come array associativo. Di seguito viene definita una

---

serie di proprietà in someObj. Infine si visualizzano su pannello di output due proprietà utilizzando in un caso l'operatore punto e nell'altro la sintassi di accesso agli array.

3. Selezionare Controllo > Prova filmato per provare il codice ActionScript.

La cosa rilevante messa in evidenza da questo esempio è che posso accedere alle proprietà di qualsiasi oggetto usando indifferentemente la sintassi del punto (someObj.myColor) e la sintassi degli array (someObj['myColor']) utilizzando quindi il nome della proprietà in formato stringa. Come vedremo questa caratteristica viene spesso usata nel codice che manipola gli oggetti.

## Date

La classe Date consente di recuperare i valori relativi alla data e all'ora del tempo universale (UTC) o del sistema operativo su cui è in esecuzione Flash Player. Per chiamare i metodi della classe Date, è prima necessario creare un oggetto Date mediante la funzione di costruzione della classe Date.

La funzione di costruzione Date() può avere diverse sintassi.

Posso passare alla funzione una data del passato o del futuro. Il questo caso Date() richiede da due a sette parametri (anno, mese, giorno, ora, minuti, secondi , millisecondi).

In alternativa, posso costruire un oggetto Date passando un singolo parametro che rappresenta il numero di millisecondi trascorsi dal 1 gennaio 1970 alle 0:00:000 GMT.

Se, infine, non specifico alcun parametro all'oggetto data Date() vengono assegnate la data e l'ora correnti.

Alcune esempi:

```
var d1:Date = new Date();  
var d3:Date = new Date(2000, 0, 1);  
var d4:Date = new Date(65, 2, 6, 9, 30, 15, 0);  
var d5:Date = new Date(-14159025000);
```

Nella prima riga di codice, un oggetto Date viene impostato sull'ora e la data correnti.

Nella seconda riga, viene creato un oggetto Date a cui vengono passati i parametri year, month e date, con la data e l'ora corrispondenti alle 0:00:00 GMT del 1 gennaio 2000.

---



Nella terza riga, viene creato un oggetto Date a cui vengono passati i parametri year, month e date, con la data e l'ora corrispondenti alle 09:30:15 GMT (+ 0 millisecondi) del 6 marzo 1965. Poiché il parametro year è specificato sotto forma di intero a due cifre, viene interpretato come 1965.

Nella quarta riga, viene passato solo un parametro, ovvero un valore temporale che rappresenta il numero di millisecondi prima o dopo le 0:00:00 GMT del 1 gennaio 1970; dal momento che il valore è negativo, rappresenta un tempo *precedente* alle 0:00:00 GMT del 1 gennaio 1970 e in questo caso l'ora e la data sono le seguenti: 02:56:15 GMT del 21 luglio 1969.

Vediamo ora come si utilizza l'oggetto Date. Nell'empio che segue realizzeremo un semplice orologio-datario.

4. Creare un nuovo documento Flash e salvarlo come Date\_1 fla.
5. Creare sullo stage un campo di testo dinamico. Scegliere come font del campo \_sans e dargli il nome messaggio\_txt.
6. Immettere il codice ActionScript seguente nel fotogramma 1 della linea temporale:

```
stop();
//semplice orologio-calendario perpetuo
//utilizza setInterval()
//creo un array mesi che contiene i nomi dei mesi
var mesi:Array = new Array("gennaio", "febbraio", "marzo",
    "aprile", "maggio", "giugno", "luglio", "agosto",
    "settembre", "ottobre", "novembre", "dicembre");

//e un array giorni che contiene i nomi dei giorni della
    settimana
var giorni:Array = new Array("domenica", "lunedì", "martedì",
    "mercoledì", "giovedì", "venerdì", "sabato");

//scrivo una semplice funzione che aggiunge uno "0"
//davanti ad un numero se è composto da una sola cifra
//e lo trasforma in stringa
function zeroPrima(n:Number):String {
    var s:String = n.toString();
    if (s.length == 1) {
        s = "0" + s;
    }
    return (s);
}

//definisco la funzione aggiornaOrologio() che non ritorna
//alcun valore e che aggiorna il contenuto del campo di
//testo messaggio_txt sulla base di quanto restituito
//dall'oggetto now
```

```

function aggiornaOrologio ():Void {
    //creo un oggetto date che contiene data e ora corrente
    var now:Date = new Date();

    var giorno_sett:String = giorni[now.getDay()];
    var giorno:Number = now.getDate();
    var mese:String = mesi[now.getMonth()];
    var anno:Number = now.getFullYear();
    var ora:String = zeroPrima(now.getHours());
    var minuti:String = zeroPrima(now.getMinutes());
    var secondi:String = zeroPrima(now.getSeconds());
    messaggio_txt.text = giorno_sett + " " + giorno + " " +
    mese + " " + anno + " " + ora + ":" + minuti + ":" +
    secondi;
}
// la funzione aggiornaOrologio viene eseguita una prima
// volta appena il filmato viene eseguito

aggiornaOrologio();

// poi utilizzando il timer del computer la funzione
// aggiornaOrologio viene eseguita ogni 100 millisecondi

setInterval(aggiornaOrologio, 100);

```

Il codice crea due array che contengono rispettivamente i nomi dei mesi e dei giorni della settimana in lingua italiana. Poi definisce una funzione `zeroPrima()` che trasforma un numero in stringa formattandolo con uno zero iniziale se è composto da una sola cifra. La funzione serve per formattare in maniera più leggibile ore, minuti e secondi del mio orologio. Definisce poi una funzione `aggiornaOrologio()` che crea un oggetto `Date` (`now`) che contiene l'ora e la data correnti. Usando vari metodi della classe **Date** estrae dall'oggetto `now` le informazioni su giorno, mese, anno, giorno della settimana, ora, minuti e secondi. Utilizzando la funzione `zeroPrima()` e i due array le informazioni vengono trasformate in un formato leggibile e quindi usate per aggiornare il campo di testo. Chiamando la funzione `setInterval()` (funzione incorporata nel linguaggio) si imposta un timer del computer in modo che la funzione `aggiornaOrologio()` venga eseguita dieci volte al secondo.

7. Selezionare Controllo > Prova filmato.

`Date` mi può servire anche per fare operazioni sulle date. Ad esempio posso chiedermi quanti giorni sono trascorsi dal 23 dicembre 1981 ad oggi.

1. Creare un nuovo documento Flash e salvarlo come `Date_2 fla`.
2. Immettere il codice ActionScript seguente nel fotogramma 1 della linea temporale:

```

//differenza tra due date
var data_1:Date = new Date(1981,23,11);

```

```
var oggi:Date = new Date();  
  
var diff:Number = oggi.getTime() - data_1.getTime();  
var giorni_trascorsi:Number = Math.round(diff/(3600000*24));  
trace(giorni_trascorsi);
```

Il codice crea due oggetti Date. Il primo contiene il 23 dicembre 1981 (i mesi sono numerati a partire da 0 e quindi dicembre vale 11). Il secondo la data di oggi. Usando il metodo `getTime()` si ottiene il numero di millisecondi trascorsi prima o dopo il 1 gennaio 1970 per entrambe le date. E' possibile quindi calcolare facilmente l'intervallo tra le due date in millisecondi e quindi confertirlo nell'unità di misura che preferisco.

3. Selezionare Controllo > Prova filmato. Sul pannello di output appare il numero di giorni trascorsi dal 23 dicembre 1981.