# BREVE RIEPILOGO

# INFORMATICA

- disciplina che studia l'elaborazione automatica di informazioni.
- Codifica
- Informazione => dato

# DIGITALE

- Digitale è un'informazione codificata cioè trasformata in sequenze di 0 e 1.
- Codifica
- Informazione => dato

# DIMENSIONI

- Un insieme di informazioni può essere:
  - **finito** o **infinito**
  - **continuo** o **discreto**
- Un'informazione codificata è sempre finita e discreta.

# COMPRESSIONE

- Le informazioni sono spesso ridondanti.
- Con la codifica la ridondanza può aumentare
- La ridondanza consente la compressione

# COMPRESSIONE

- Compressione **lossless**
  - sfrutta la ridondanza
  - dall'informazione compressa è sempre possibile risalire all'informazione originale
- Compressione **lossy**
  - è specifica per determinati tipi di informazioni
  - sfrutta meccanismi percettivi
  - dall'informazione compressa non è possibile risalire all'informazione originale

# 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.

# ALGORITMO

- Si può rappresentare con:
  - pseudolinguaggio
  - diagramma di flusso

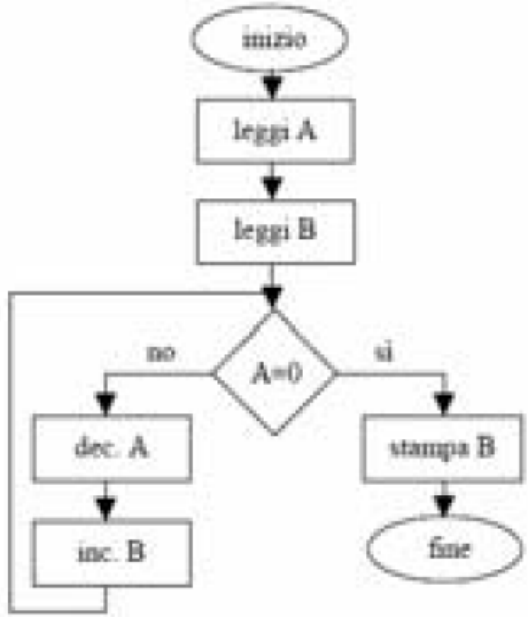
# PROBLEMA:

sommare due numeri naturali utilizzando solo incrementi e decrementi

## Pseudolinguaggio

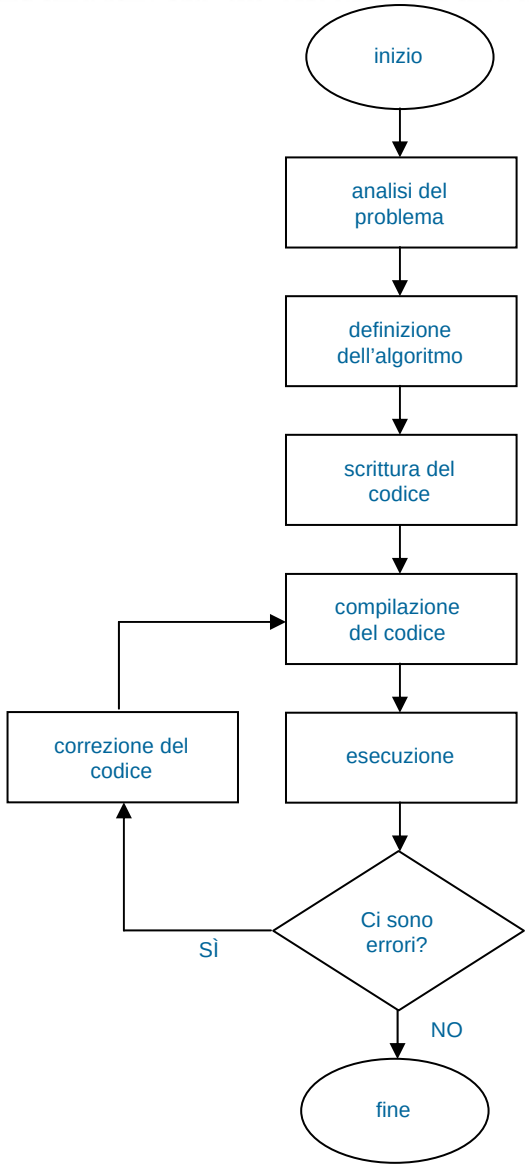
1. Leggi A
2. Leggi B
3. Se A=0 vai all'istruzione 7
4. Decrementa A
5. Incrementa B
6. Vai all'istruzione 3
7. Stampa B

## Diagramma di flusso



# IL PROCESSO

Se utilizziamo un diagramma di flusso per descrivere il processo di creazione di un'applicazione otterremo lo schema a fianco.

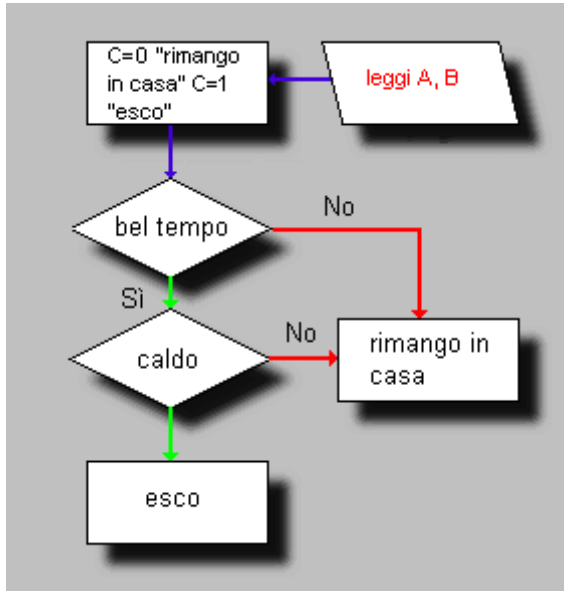


# Leggibilità significa:

- Progettare con chiarezza
- Scrivere codice con chiarezza

# Formalizzazione

- *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*

# 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 |

NOT - Negazione

| | |
|---|---|
| NOT falso | risultato vero |
| NOT vero | risultato falso |

OR - Disgiunzione

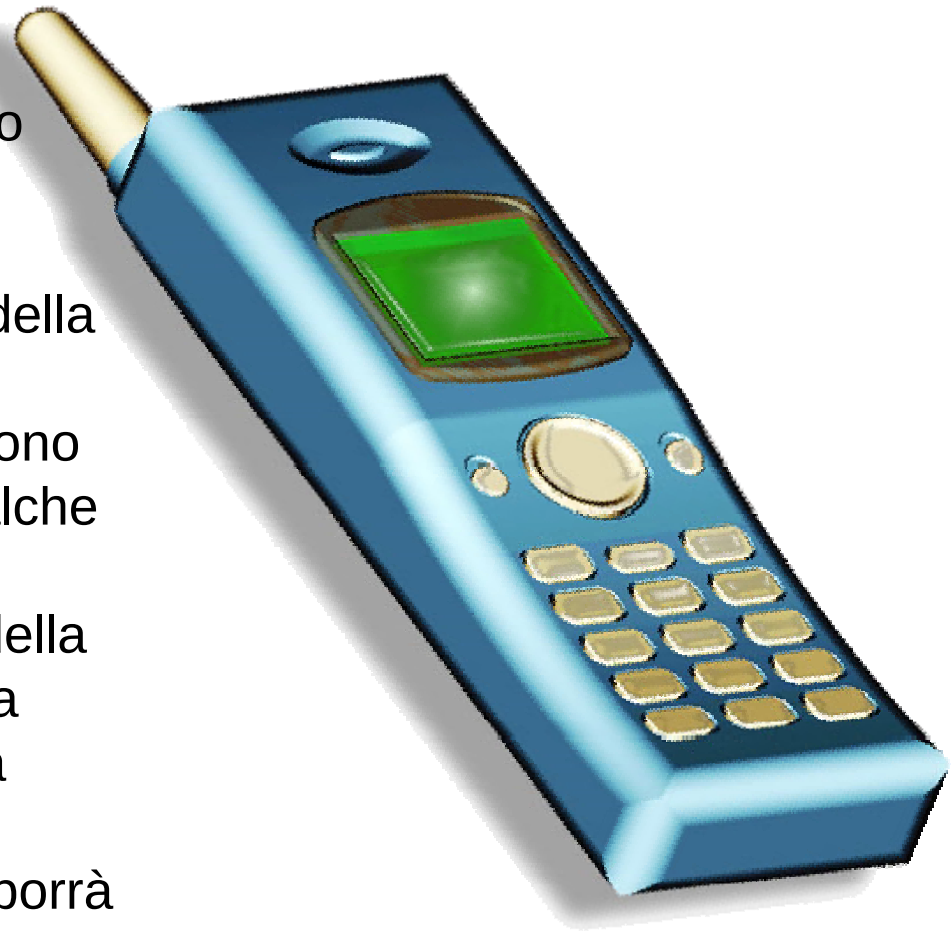
| | |
|---|---|
| falso OR falso | risultato falso |
| falso OR vero | risultato vero |
| vero OR falso | risultato vero |
| vero AND vero | risultato vero |

# Elementi di un linguaggio

- Le unità semantiche di base di un linguaggio sono:
  - ***Parole chiave***
  - ***Operatori e separatori***
  - ***Letterali*** (o ***Costanti***)
  - ***Nomi*** (o ***Identificatori***)

# Variabili

Pensiamo 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.



# Variabili

- 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.
- Ho la possibilità di usare simboli mnemonici al posto di numeri e stringhe di grande entità o difficili da ricordare.
- Ho la possibilità di usare il nome della variabile al posto del suo valore per eseguirvi sopra delle operazioni, e generalizzare l'elaborazione.

# Esempio

- Questo un programma scritto in pseudolinguaggio calcola il quadrato di un numero inserito dall'utente e lo mostra sul video.

```
scrivi sullo schermo "Ciao Inserisci un numero";
A = -numero inserito da tastiera-;
B = A * A;
scrivi sullo schermo "Il quadrato di " A " è " B;

/* A e B sono variabili */
```

# 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.
- La dichiarazione è un comando che comunica al compilatore che un determinato nome è il nome di una variabile e che quella variabile conterrà un determinato tipo di dati.

# Tipi primitivi

- I tipi primitivi sono i tipi quelli fissati dalle specifiche del linguaggio.
- Posso manipolare i tipi primitivi utilizzando gli operatori.
- Le variabili contengono  completamente un valore di un tipo primitivo.
- In ActionScript i tipi primitivi sono **Number**, Boolean e String.

# 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**.

# Number

- Questo tipo di dati può rappresentare numeri interi, numeri interi senza segno e numeri a virgola mobile.
- Per memorizzare un numero a virgola mobile, includere una punto decimale nel numero; senza il punto il numero viene memorizzato come numero intero e quindi i risultati delle operazioni vengono arrotondate al numero intero più vicino.

# 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.

# Diciarazioni di varibili

- Per dichiarare una varibile in ActionScrit si usa la parola riservata **var** seguita dal nome della variabile, dai due punti e dal tipo:

```
var pippo:String;
```

- Opzionalmente si può assegnare un valore alla variabile all'atto della dichiarazione (inizializzazione):

```
var pippo:String = "Hello World";
```

# Dichiarazione di variabili di tipi primitivi

```
//dichiarazioni di variabili in actionscript
/* a può contenere solo un numero, s una
  stringa k true o false */
var a:Number;
var s:String;
var k:Boolean;
/* per b e messaggio oltre a dichiarare il
  tipo viene Impostato un valore iniziale */
var b:Number = 1;

var messaggio:String = “Ciao a tutti”;
```

# Tipi derivati o complessi

- Per rappresentare dati 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.
- Per i tipi complessi la variabile contiene il *puntatore* cioè il numero della casella, l'indirizzo in cui il dato è memorizzato sul computer.
- Nei linguaggio orientati agli oggetti il concetto di **tipo** e il concetto di classe coincidono.

# PROGRAMMAZIONE CONDIZIONALE

# Sintassi dell'istruzione if

- L'istruzione if consente di tradurre in un linguaggio di programmazione i ragionamenti fatti parlando della logica Booleana.
- L'istruzione if può avere due forme:

  **if** ( espressione ) blocco di istruzioni

  **if** ( espressione ) blocco di istruzioni **else** blocco di istruzioni
- 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 vera si esegue l'istruzione seguente, altrimenti si esegue l'istruzione subito dopo la parola chiave **else**.
- 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.

# Esempio in pseudocodice

```
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";
}
```

# Esempio in ActionScript

```
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!!!";
}
```

# PROGRAMMAZIONE ITERATIVA

# La programmazione Iterativa

- **Programmazione procedurale:**
  - viene eseguita un'istruzione dopo l'altra fino a che non si incontra l'istruzione di fine programma.
- **Programmazione iterativa:**
  - un'istruzione (o una serie di istruzioni) vengo eseguite continuamente, fino a quando non sopraggiungono delle condizioni che fanno terminare il ciclo.

# while, do e for

- In quasi tutti i linguaggi di programmazione si usano tre costrutti per ottenere l'iterazione:
  - **while**
  - **do**
  - **for**
- La funzione è la stessa con modalità leggermente diverse.

# while

- L'istruzione **while** viene schematizzata come segue:

```
while ( condizione )
        blocco istruzioni;
```

- Con questa istruzione viene prima valutata l’espressione  <condizione>, se l'espressione risulta vera viene eseguito <blocco istruzioni> e il blocco **while** viene ripetuto, altrimenti si esce dal ciclo e si procede con il resto del programma.

# do

- L'istruzione **do** può essere considerato una variante dell'istruzione **while** ed è strutturato nella maniera seguente:

```
do
  blocco istruzioni
  while ( condizione )
```

- Prima di tutto viene eseguito il blocco di istruzioni 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.

# for

- Il **for** inizializza una variabile, pone una condizione e poi modifica (normalmente incrementa o decrementa) la variabile iniziale.

```
for (inzializzazione; condizione; modifica)
  blocco istruzioni;
```

- Il codice  <blocco istruzioni> viene eseguito fino a che l'espressione <condizione> risulta vera, poi si passa la all'istruzione successiva al **for**.

# for e while

- Spesso un ciclo **for** può essere trasformato in un ciclo **while** di questo tipo:

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

# for .. in

- Un ciclo **for..in** consente di eseguire iterazioni scorrendo gli elementi di un ogetto (un clip filmato, un oggetto generico, o un array).
- La struttura del comando è:

```
for variabile in oggetto
    blocco istruzioni;
```

- Il ciclo prende in esame tutti gli elementi presenti in <oggetto>. Ad ogni ciclo <variabile> assume il nome dell'elemento preso in esame e <blocco istruzioni> viene eseguito.

# La successione di Fibonacci

- 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 $F_n := F_{n-1} + F_{n-2}$.

# La successione di Fibonacci

```
1.      numero n1 = 0;                  //contiene il numero n–2 della serie
2.      numero n2 = 1;                  //contiene il numero n–1 della serie
3.      numero n3;                      //numero di Fibonacci che viene calcolato
4.      numero contatore = 0;           //conta i numeri di fibonacci creati
5.      stringa elenco = "";            //Stringa di testo che conterrà l'elenco dei numeri
6.      scrivi sullo schermo "Inserisci un numero maggiore di 2";
7.      numero A = - numero inserito dall'utente –
8.      if (A maggiore o uguale 2) {
9.          elenco = "0, 1, ";    //se il numero inserito è maggiore di due elenco conterrà
10.                                       //all'inizio i primi due numeri della serie
11.         A = A - 2;
12.     }

13.     n3 = n2 + n1;               //il numero di viene calcolato sommando i due numeri
                                    //precedenti
14.     elenco = elenco + n3 + ", ";        //a elenco viene aggiunto il numero di Fibonacci
                                            // così ottenuto più una virgola e uno spazio

15.     n1 = n2;       //in n1 e n2 vengono ora memorizzati gli ultimi due numeri della serie
16.     n2 = n3;
17.     incrementa contatore;
18.     if (contatore < A) goto 13;         // il ciclo continua fino a che non sono stati
                                            // creati tutti numeri richiesti dell'utente.
19.     scrivi sullo schermo elenco;        // la stringa così ottenute viene scritta sullo
                                            // schermo
```

# Versione ActionScript

```
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) {
    elenco = "0, 1, ";           //se il numero inserito è maggiore di u uguale a 2 elenco
                                 //conterrà all'inizio i primi due elementi della serie
}
for (var i:Number = 0; i < (A-2); i++) {
    n3 = n2 + n1;     //il numero di viene calcolato sommando i due numeri precedenti
    n1 = n2;          //in n1 e n2 vengono ora memorizzati gli ultimi due numeri della serie
    n2 = n3;
    elenco = elenco + n3 + ", ";  //a elenco viene aggiunto il numero di Fibonacci così
                                  // ottenuto Più e una virgola e uno spazio
}
// 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
```

# FUNZIONI E METODI

# COSA È UNA FUNZIONE

- Una funzione (o procedura o metodo) è una costrutto presente in tutti i linguaggi di programmazione che consente di associare un gruppo di comandi ad un identificatore.
- Quando nel programma scriverò l'identificatore saranno eseguiti tutti i comandi che compongona la funzione

# UTILITÀ DELLE FUNZIONI

- L'uso di funzioni ha due vantaggi:
  - evitare di scrivere codice ripetitivo
  - rendere il mio programma modulare facilitando così modifiche e correzioni.

# IN ACTION SCRIPT

- Le ***funzioni*** sono blocchi di codice ***ActionScript*** riutilizzabili in qualsiasi punto di un file SWF
- I ***metodi*** sono semplicemente funzioni che si trovano all'interno di una definizione di **classe** *ActionScript*.

# DICHIARAZIONE E DEFINIZIONE

- Una funzione deve essere **dichiarata** e **definita**;
  - cioè vanno specificati i tipi di ingresso e di uscita sui quali la funzione andrà a compiere le proprie operazioni (**DICHIARAZIONE**)
  - e successivamente dovremo scrivere il **corpo** della funzione vera e propria (**DEFINIZIONE**).
  - all'interno del corpo della funzione potrò definire un **valore di ritorno**.

# ESEMPIO

```
function somma(n1:Number, n2:Number):Number{
   return (n1 + n2);
}
```

- Questo codice dichiara la funzione somma che accetta due parametri che devono essere numeri e restituisce un numero.
- La funzione viene poi definita dal blocco di codice tra le due parentesi graffe. Il comando fa che la funzioni ritorni la somma dei due numeri passati come parametri. Se scrivo:

```
var a:Number;
a = somma(5, 7);
```

  a conterrà 12.

# FUNZIONI INCORPORATE

- Nel linguaggio *ActionScript* sono incorporate numerose 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.
- 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***.

# ESEMPIO

- 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:
  - ***funzioni con nome***
  - ***funzioni anonime***

# SCRITTURA DI FUNZIONI CON NOME

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

- 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.

# ESEMPIO

```
stop();
//semplice timer che misura il tempo trascoso
//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 utilizzazndo il timer del computer la funzione
// my_timer viene eseguita ogni 30 millisecondi
setInterval(my_timer, 30);
```

# SCRITTURA DI FUNZIONI ANONIME

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

- nomevaribile 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.

# ESEMPIO

```
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);
```

# SCRITTURA DI FUNZIONI ANONIME

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

- nomevaribile 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.

# ESEMPIO

```
stop();
//uso di una funzione anonima per gestire eventi
//creo un'istanza della classe sound
var mio_suono:Sound = new Sound();
//la variabile mio_souno 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_souno
mio_suono.loadSound("Round.mp3", false);
```

# PASSAGGIO DI PARAMETRI

- 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.

# ESEMPIO

```
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(somma(1, 4, 6, 8)); // 11
```

# RESTITUZIONE DI VALORI

- 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 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.