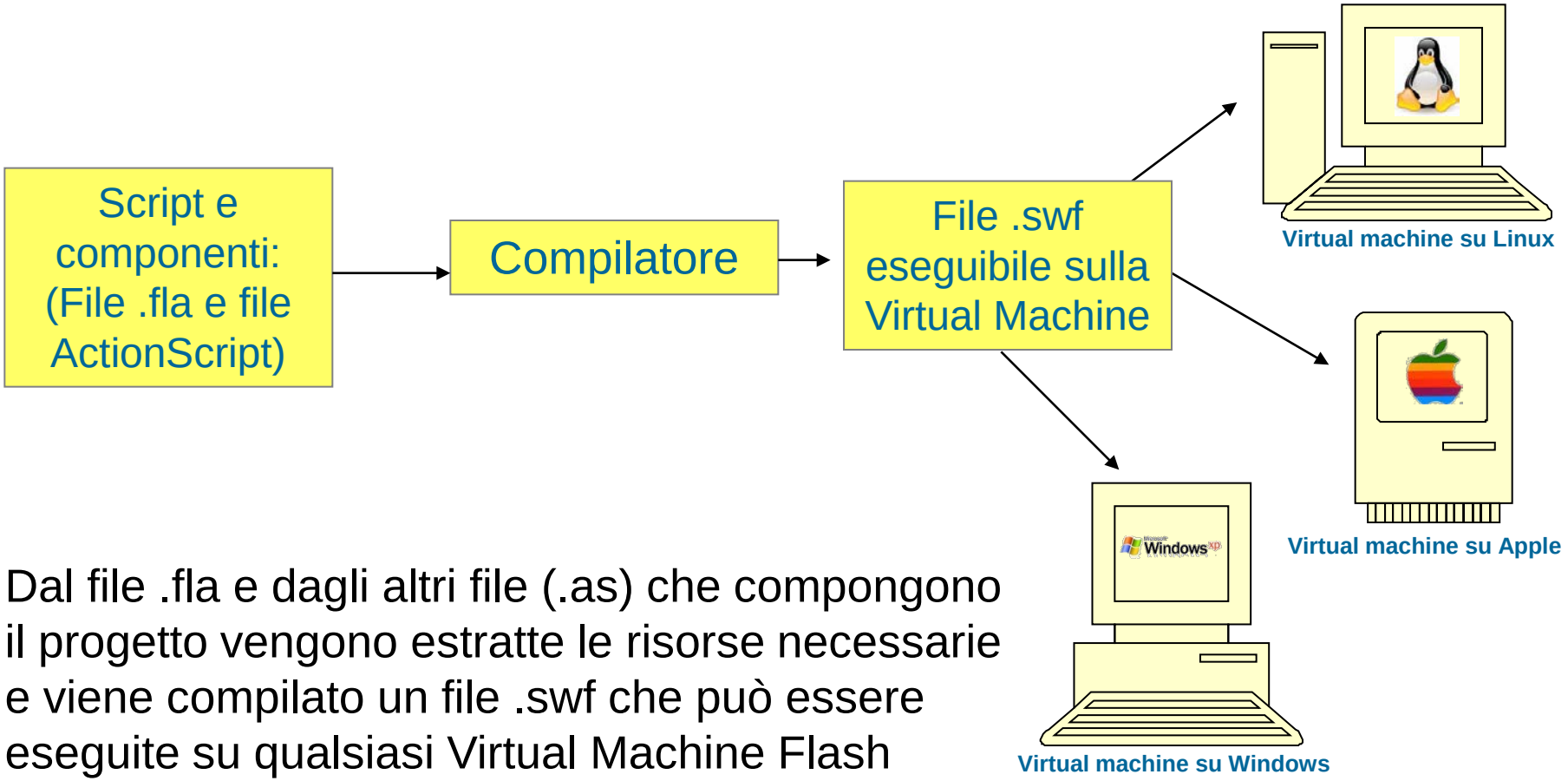
RIPASSO

ACTIONSCRIPT È UN LINGUAGGIO COMPILATO



Dal file .fla e dagli altri file (.as) che compongono il progetto vengono estratte le risorse necessarie e viene compilato un file .swf che può essere eseguite su qualsiasi Virtual Machine Flash

ELEMENTI DI UN LINGUAGGIO

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

PAROLE CHIAVE

and, break, case, catch, class, const, continue, default, delete, do, dynamic, else, extends, finally, for, function, get, if, implements, import, in, instanceof, interface, intrinsic, new, not, or, package, private, public, return, set, static, switch, this, throw, try, typeof, var, void, while, with

OPERATORI

- Gli operatori sono token composti di uno o più caratteri speciali che servono a controllare il flusso delle operazioni che dobbiamo eseguire o a costruire ***espressioni***
- Operatori usati sia in *ActionScript* che in *JAVA*:

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

SEPARATORI

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

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

COMMENTI

- Delimitatori di riga: tutto cio che segue il contrassegno di commento fino alla fine della riga non viene compilato. Esempi:

```
//
```

- Delimitatori di inizio e fine: tutto ciò compreso tra il contrassegno di inizio e il contrassegno di fine non viene compilato.

```
/*    */
```

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.
- La sintassi con cui le costanti sono descritte dipende dal tipo di dati che rappresentano.

COSTANTI

```
//costante di tipo Number
3.141592653589793
//costante di tipo string
“Il quadrato di “
//costante di tipo Array
[ "Gennaio", "Febbraio", "Marzo",
  "Aprile","Maggio","Giugno","Luglio",
  "Agosto", "Settembre", "Ottobre",
  "Novembre", "Dicembre“ ]
```

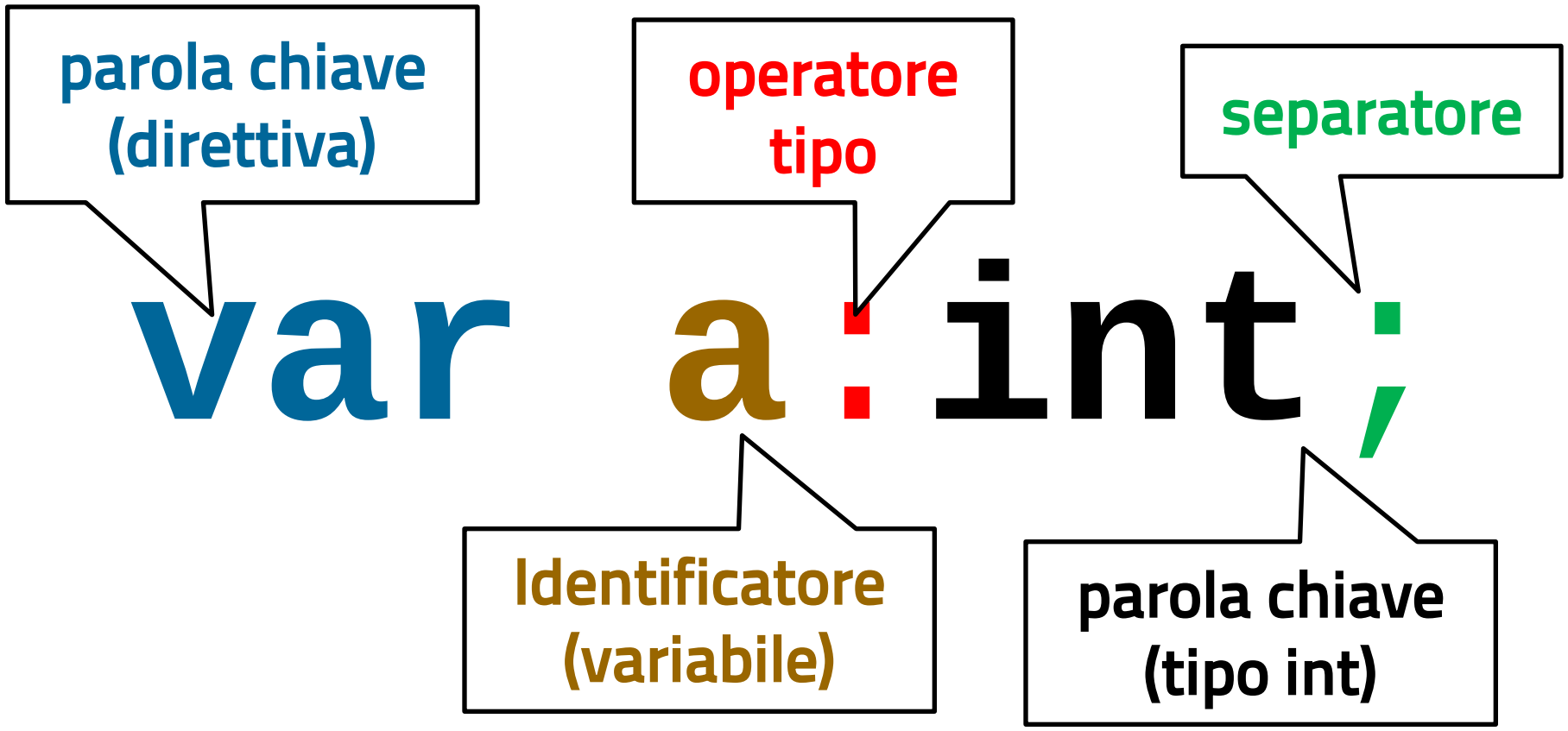
VARIABILI

- Una variabile è composta da due elementi: il suo **nome** e il suo **valore**;
- 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.

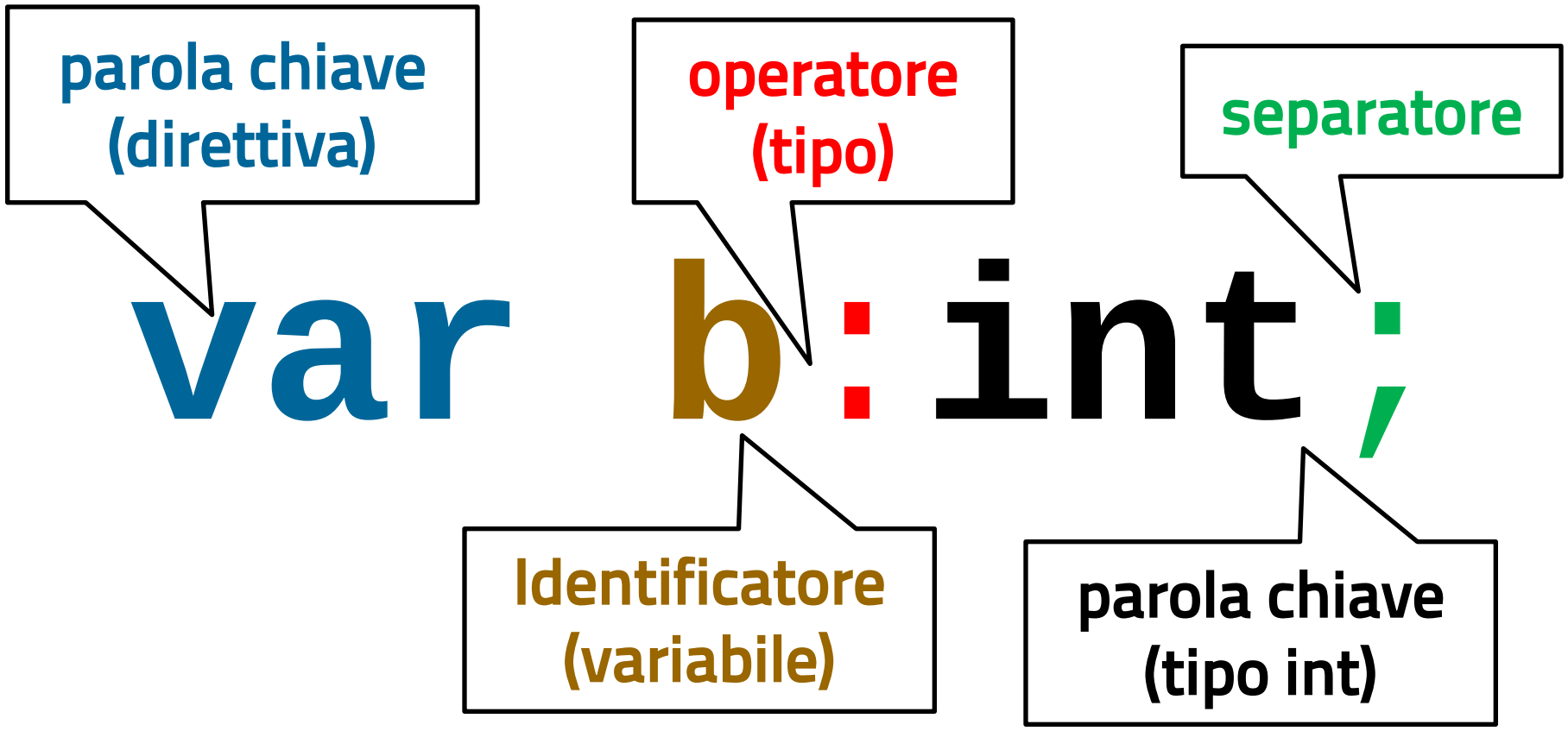
ISTRUZIONI

```
var a:int;
var b:int;
a = parseInt(numero_txt.text);
b = a*a;
messaggio_txt.text = "Il
  quadrato di " + a + " è “ + b;
```

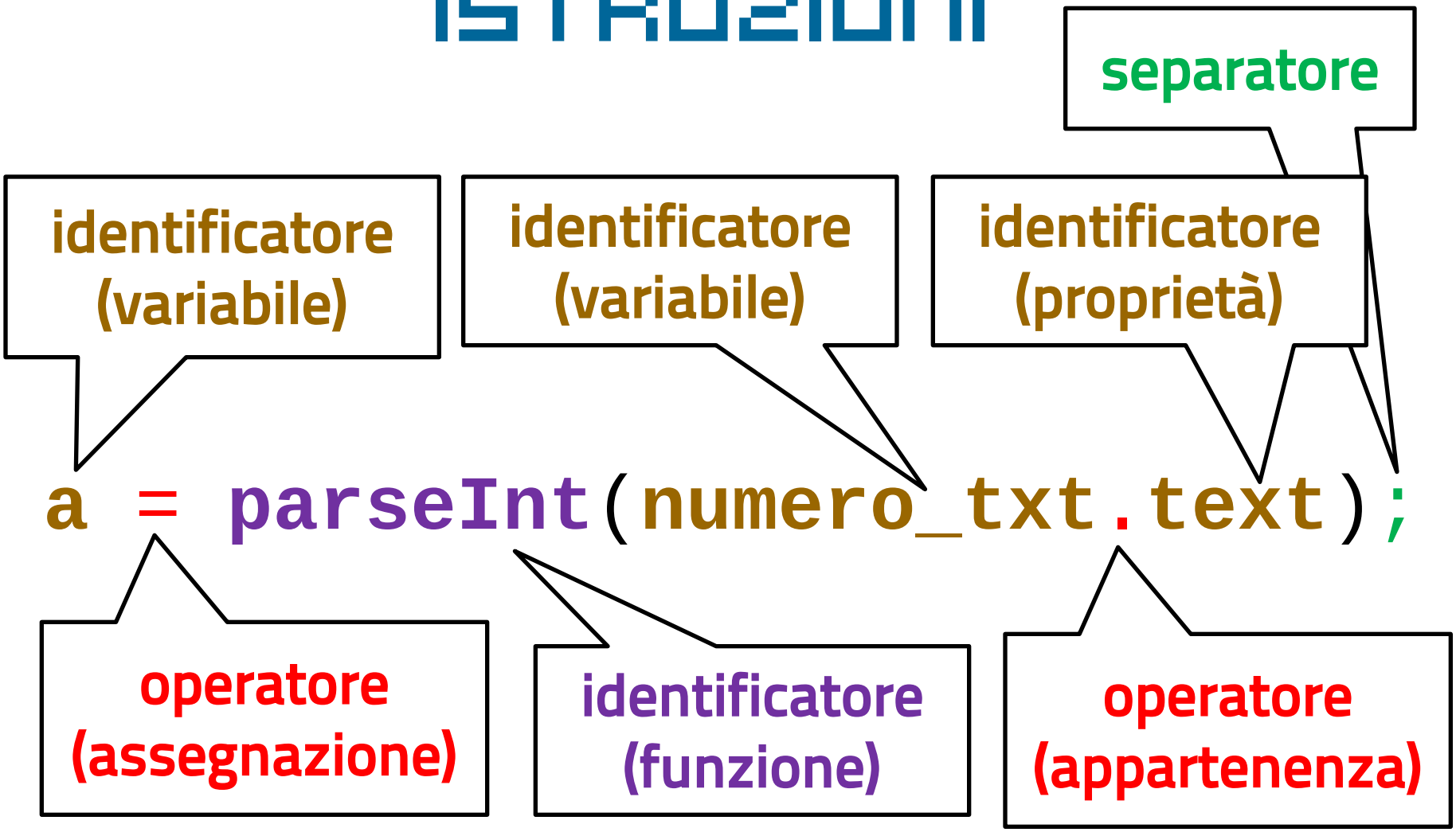
ISTRUZIONI



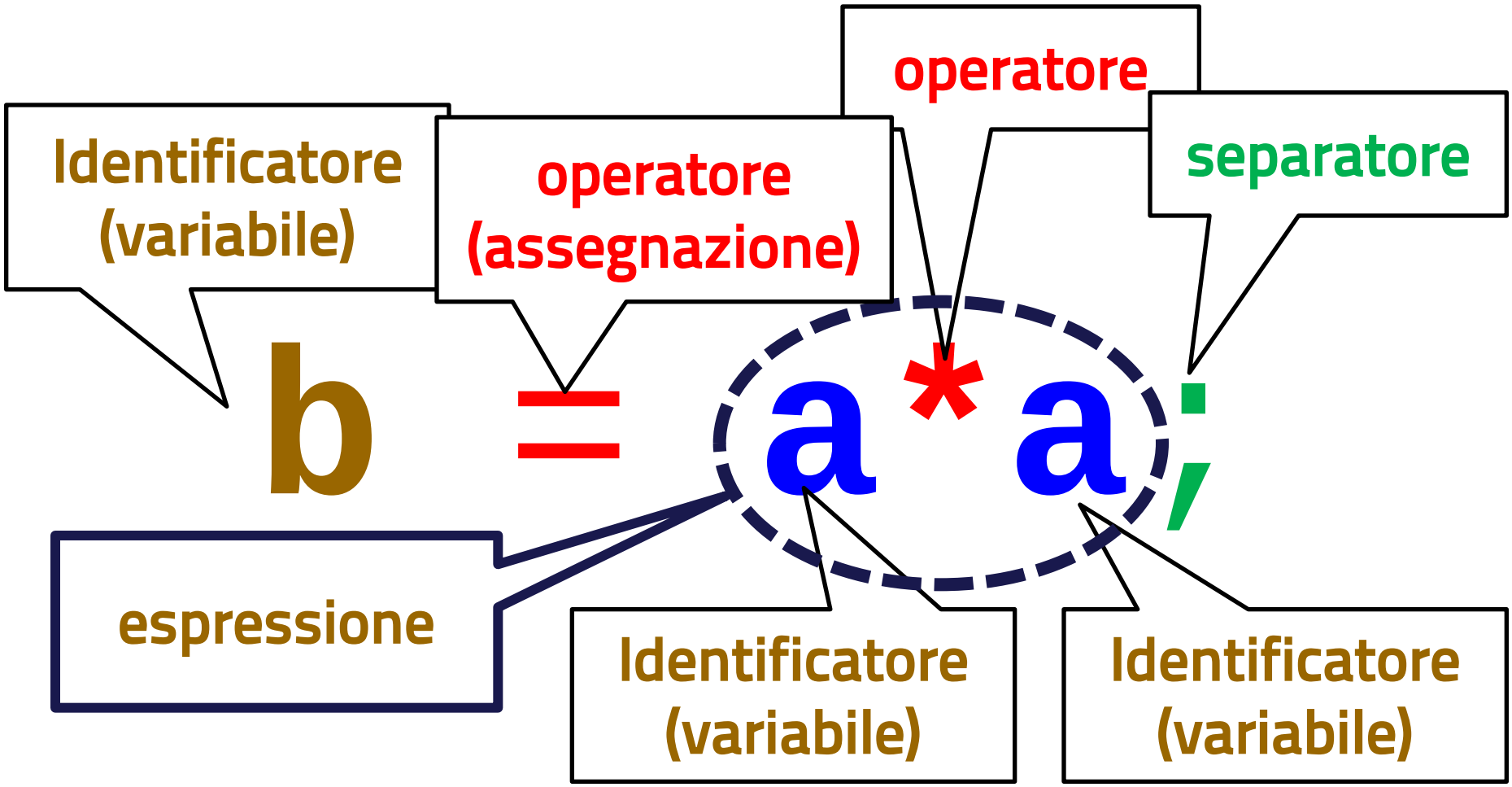
ISTRUZIONI



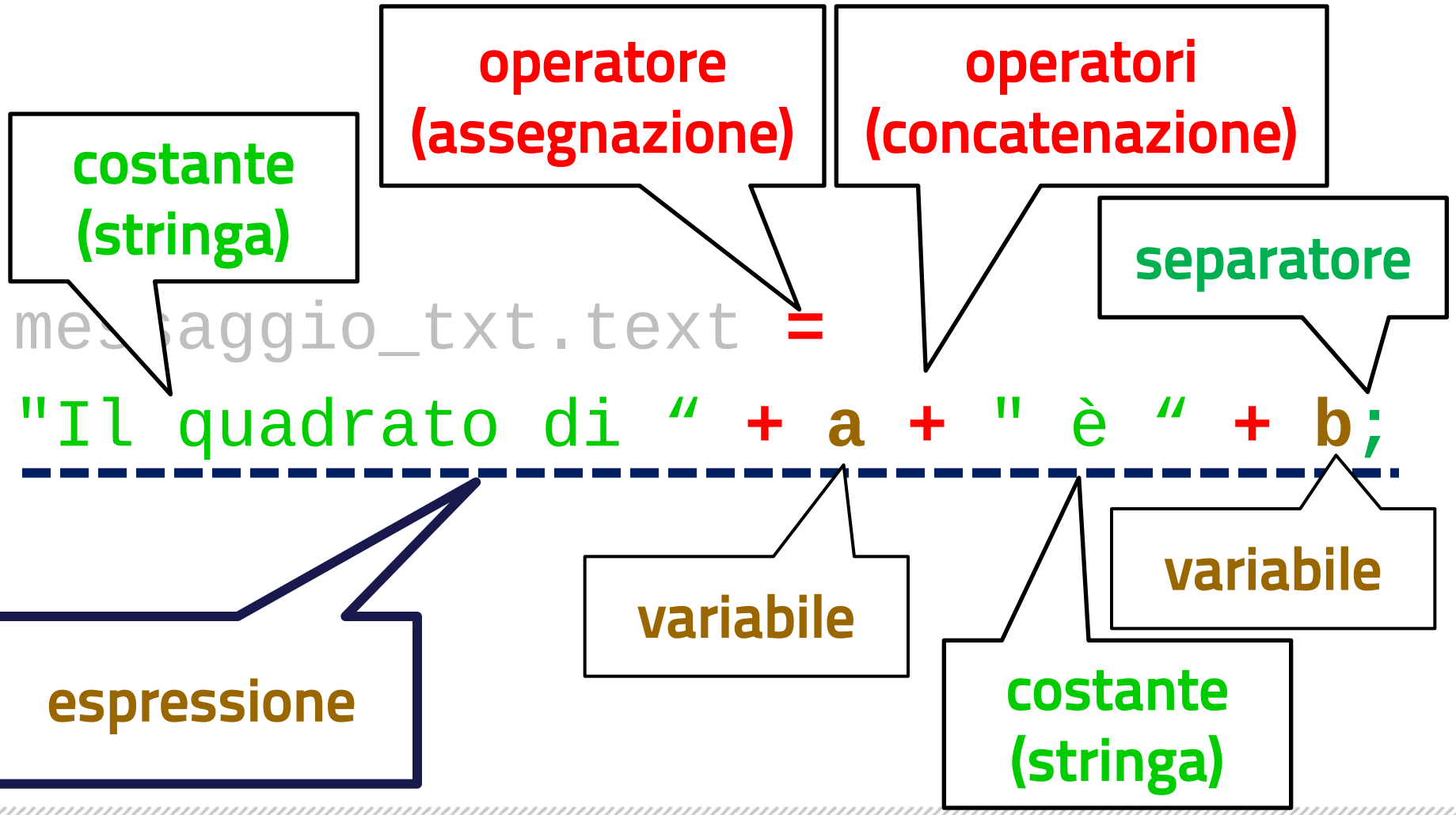
ISTRUZIONI



ISTRUZIONI



ISTRUZIONI



TIPI PRIMITIVI

- I tipi primitivi sono tipi di dato non complessi fissati dalle specifiche del linguaggio.
  - Posso manipolare i tipi primitivi utilizzando gli operatori.
  - Le variabili contengono direttamente il dato.
- In ActionScript i tipi primitivi sono **int**, **uint**, **Number**, **Boolean**, **String** a questi si aggiungo due tipi *speciali*: **void** e **null**.

TIPI DERIVATI O COMPLESSI

- Per rappresentare dati complessi 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 linguaggi orientati agli oggetti il concetto di **tipo** e il concetto di **classe** coincidono.

DICHIARAZIONE DI VARIABILI

- 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 conterrà 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” ;
```

ESEMPI DI TIPI COMPLESSI

- **Array** rappresenta un tabella di valori. Posso recuperare un valore nella tabella utilizzando l'indice.
- **Object** rappresenta un variabile strutturata che organizza un dato complesso.
- **MovieClip** rappresenta un clip filmato.
- **TextField** rappresenta un campo di testo.
- **MioTipo** un tipo creato dal programmatore.

ARRAY

- Un **Array** è un elenco di elementi recuperabile attraverso un indice.
- Non occorre che gli elementi dell'array abbiano lo stesso tipo di dati. È possibile inserire numeri, dati, stringhe, oggetti, altri array.

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

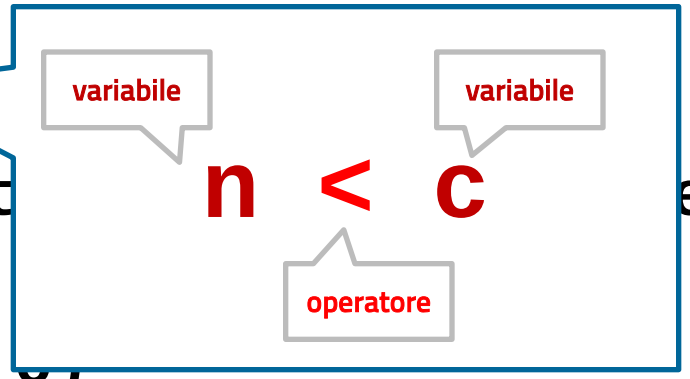
  if (espressione) blocco di istruzioni else blocco di istruzioni
- L'espressione che compare dopo la parola chiave if deve essere di tipo logico, se la condizione risulta vera viene eseguita l'istruzione subito seguente; nel secondo caso, invece, se la condizione risulta vera si esegue l'istruzione seguente, 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

```
var c:Number = 2.5;
var n:Number;
n = parseFloat(input_txt.text);

if (n < c)
{
     messaggio_txt.text = "Il numero " + n + " è minore di “ +  c;
}
else if (n == c)
{
     messaggio_txt.text = "Il numero " + n + " è uguale a " + c;
}
else if (n > c)
{
     messaggio_txt.text = "Il numero " + n + " è maggiore di " + c;
}
else
{
     messaggio_txt.text = "Inserisci un numero!";
}
```

ESEMPIO




```
if (n < c)
{
   messaggio_t                    ero " + n +
   " è minore
}
else if (n == c)
{
   messaggio_txt.text = "Il numero " + n +
   " è uguale a " + c;
}
else if (n > c)
{
   messaggio_txt.text = "Il numero " + n +
   " è maggiore di " + c;
}
else
{
   messaggio_txt.text = "Inserisci un numero!";
}
```

ESEMPIO

true

```
if (n < c)
{
   messaggio_txt.text = "Il numero " + n +
   " è minore di ” +  c;
}
else if (n == c)
{
   messaggio_txt.text = "Il numero " + n +
   " è uguale a " + c;
}
else if (n > c)
{
   messaggio_txt.text = "Il numero " + n +
   " è maggiore di " + c;
}
else
{
   messaggio_txt.text = "Inserisci un numero!";
}
```

ESEMPIO

```
if (n < c)
{
   messaggio_txt.text = "Il numero " + n +
   " è minore di “ +  c;
}
else if (n == c)
{
   messaggio_txt.text = "Il numero " + n +
   " è uguale a " + c;
}
else if (n > c)
{
   messaggio_txt.text = "Il numero " + n +
   " è maggiore di " + c;
}
else
{
   messaggio_txt.text = "Inserisci un numero!";
}
```

ESEMPIO

```
if (n < c)
{
   messaggio_txt.text = "Il numero " + n +
   " è minore di ” +  c;
}
else if (n == c)
{
   messaggio_txt.text = "Il numero " + n +
   " è uguale a " + c;
}
else if (n > c)
{
   messaggio_txt.text = "Il numero " + n +
   " è maggiore di " + c;
}
else
{
   messaggio_txt.text = "Inserisci un numero!";
}
```

ESEMPIO

false

```
if (n < c)
{
   messaggio_txt.text = "Il numero " + n +
   " è minore di " +  c;
}
else if (n == c)
{
   messaggio_txt.text = "Il numero " + n +
   " è uguale a " + c;
}
else if (n > c)
{
   messaggio_txt.text = "Il numero " + n +
   " è maggiore di " + c;
}
else
{
   messaggio_txt.text = "Inserisci un numero!";
}
```

ESEMPIO

```
if (n < c)
{
    messaggio_txt.text = "Il numero " + n +
    " è minore di  c;
}
else if (n == c)
{
    messaggio_txt.text = "Il numero " + n +
    " è uguale a " + c;
}
else if (n > c)
{
    messaggio_txt.text = "Il numero " + n +
    " è maggiore di " + c;
}
else
{
    messaggio_txt.text = "Inserisci un numero!";
}
```

false

ESEMPIO

```
if (n < c)
{
   messaggio_txt.text = "Il numero " + n +
   " è minore di " +  c;
}
else if (n == c)
{
   messaggio_txt.text = "Il numero " + n +
   " è uguale a c;
}
else if (n > c)
{
   messaggio_txt.text = "Il numero " + n +
   " è maggiore di " + c;
}
else
{
   messaggio_txt.text = "Inserisci un numero!";
}
```

false

ESEMPIO

```
if (n < c)
{
   messaggio_txt.text = "Il numero " + n +
   " è minore di ” +  c;
}
else if (n == c)
{
   messaggio_txt.text = "Il numero " + n +
   " è uguale a " + c;
}
else if (n > c)
{
   messaggio_txt.text = "Il numero " + n +
   " è maggiore di " + c;
}
else
{
   messaggio_txt.text = "Inserisci un numero!";
}
```

PROGRAMMAZIONE ITERATIVA

LA PROGRAMMAZIONE ITERATIVA

- **Flusso naturale del programma:**
  - 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;
}
```

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.

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.

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.

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.

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.

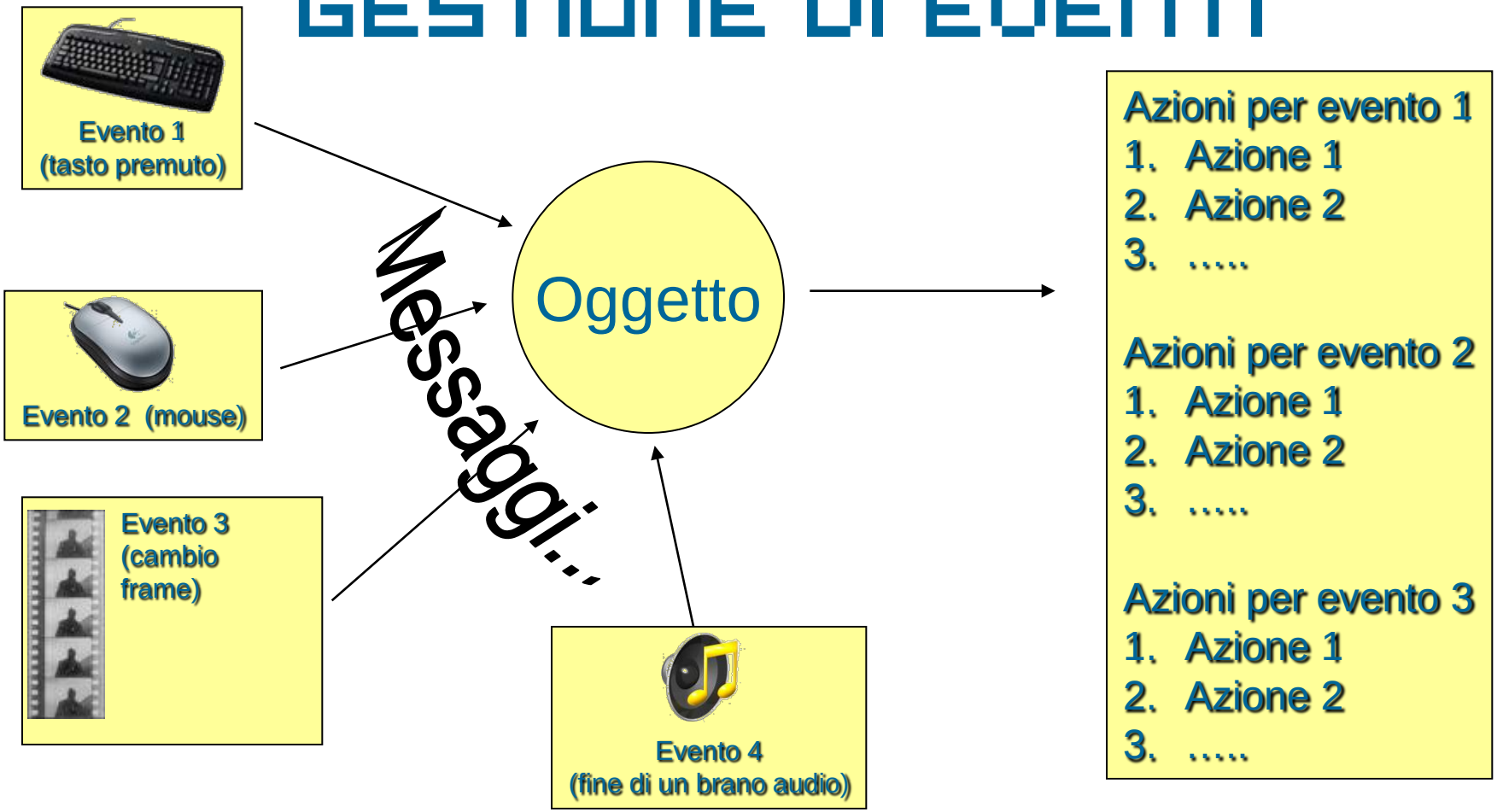
SCRITTURA DI FUNZIONI ANONIME

```
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.

FUNZIONI ED EVENTI

PROGRAMMAZIONE È GESTIONE DI EVENTI



GESTIONE DI EVENTI IN ACTIONSCRIPT

```
//uso di addEventListener
oggetto.addEventListener(nomeEvento:String,
  nomeFunzione:Function);

/* esempio */
pulsante.addEventListener(“click” ,
  calcolaQuadrato);
```