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

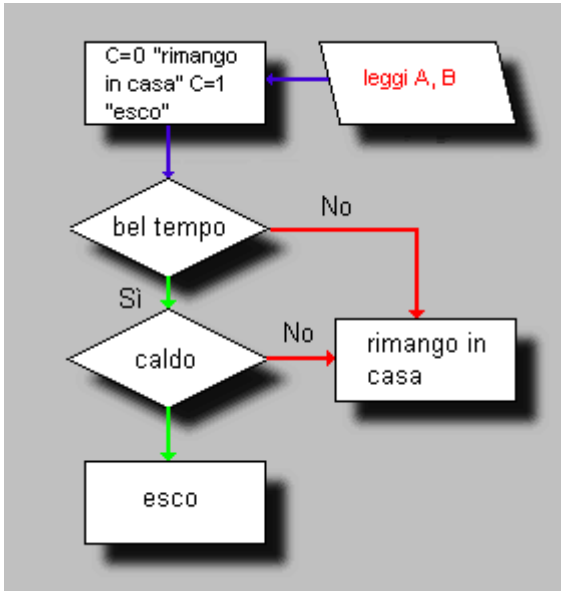
- Si può rappresentare con:
  - pseudolinguaggio
  - diagramma di flusso

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* **->** *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.

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.

```
var a:int;
var b:int;
```

VALORI E PUNTATORI

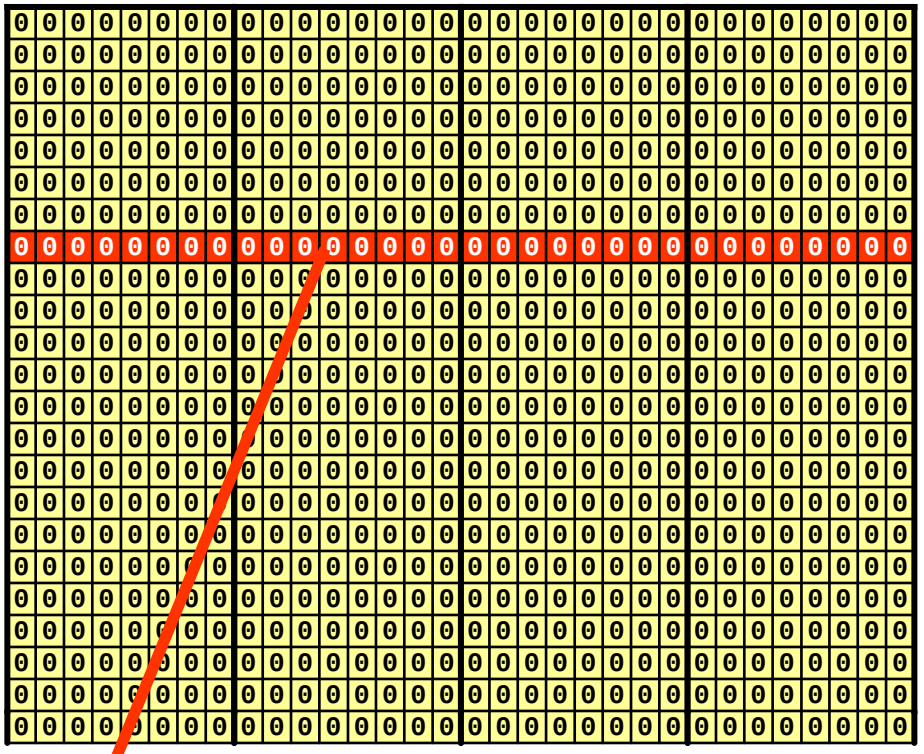
- Quando io dichiaro una variabile il compilatore riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un indirizzo anch'esso espresso in bit.
- Quando scrivo:

```
var a:int;
```

- Dico che **a** corrisponde ad una ben determinata cella di memoria composta da 32 bit.

VALORI E PUNTATORI

- Quando dichiaro una variabile il compilatore riserva uno spazio in memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un indirizzo anch'esso espresso in bit.
- Dichiarando una varibile associo la varibile a una di questa celle.

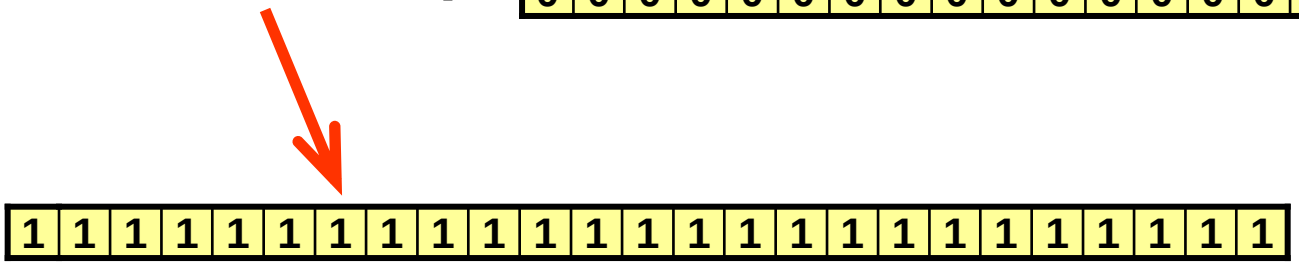


```
var a:int;
```

VALORI E PUNTATORI

- Se dico che **a** è una varabile di tipo **int** stabilisco due cose
  - Che ad a vengono riservati 32 bit
  - Che il valore contenuto nella cella viene interpretato come int (numero intero con segno)

```
a = 1000;
a = -1;
```

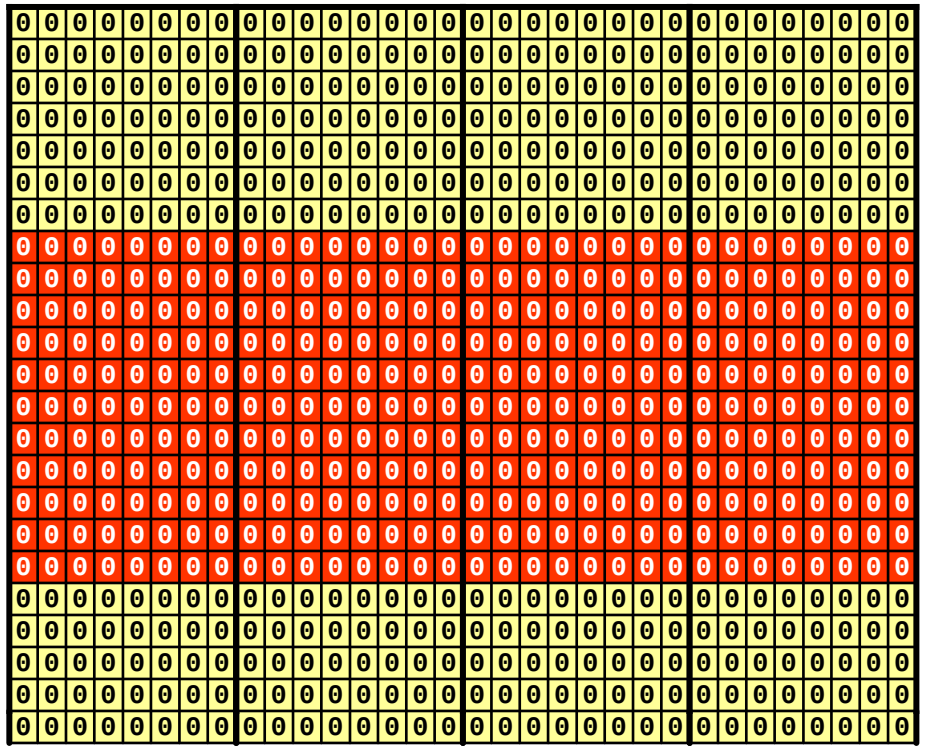


VALORI E PUNTATORI

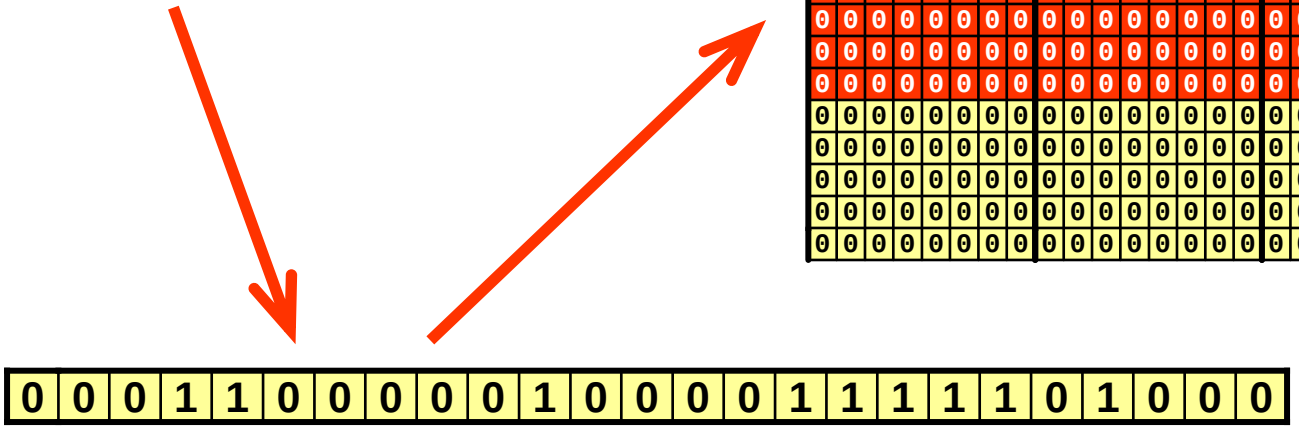
- Quando la casella che la variabile rappresenta contiene direttamente il dato si dice che la varibile **contiene un valore**.

VALORI E PUNTATORI

- Per dati più complessi la cella che la varabile rappresenta non contiene direttamente il valore ma l'indirizzo alla zona di memoria in cui il dato complesso è codificato.



```
var a:Object;
```



VALORI E PUNTATORI

- Quando la casella che la variabile rappresenta contiene l'indirizzo di memoria a partire dal quale e memorizzato il dato si dice che la variabile, **contiene un puntatore**.

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**.

int

- IL tipo di dati **int** memorizza un numero intero con segno 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 valore predefinito di una variabile di tipo **int** dichiarata ma non inizializzata è **0**.

uint

- 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.
- Il valore predefinito di una variabile di tipo **uint** dichiarata ma non inizializzata è **0**.

Number

- Il tipo **Number** 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 automatica: per forzare **Number** a rappresentare un numero decimale, bisogna includere il punto decimale (Ad esempio: **123.0**)
- Se il numero è impostato come intero i risultati delle operazioni vengono arrotondate al numero intero più vicino.

Number

- 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 == 1.7976931348623e+308**
- **Number.MIN_VALUE == 4.9406564584124e-324**
- La notazione qui utilizzata e quella scientifica (o esponenziale) ed equivale (nel primo caso) a $1.7976931348623 * 10^{308}$, cioè 179,769,313,486,231 seguito da 294 zeri.

Number

- 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 molto grandi o molto piccoli saranno molto approssimati.
- Qundo viene utilizzato per memorizzare i numeri interi il tipo **Number** è ingrado di gestire valori da **-9,007,199,254,740,992 (-2^{53})** a **9,007,199,254,740,992 ($2^{53}-1$)** compresi.
- Il valore di default di una variabile **Number** è **NaN**.

Boolean

- Il tipo di dati Boolean può avere due valori simbolici **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**.
- Astrattamente il valore boleano è rappresentato da un unico bit.

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: la costante 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.

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” ;
```

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.

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.

USO DEGLI ARRAY

- Gli array possono essere utilizzati in modi diversi.
- Il modo più semplice di creare un Array è assegnare ad una varibile di tipo Array un elenco si valori sotto forma di costante.
- 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.
- Per identificare un elemento di un Array il nome della variabile viene seguito dall’indice tra parentesi quadre.

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;
}
```

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

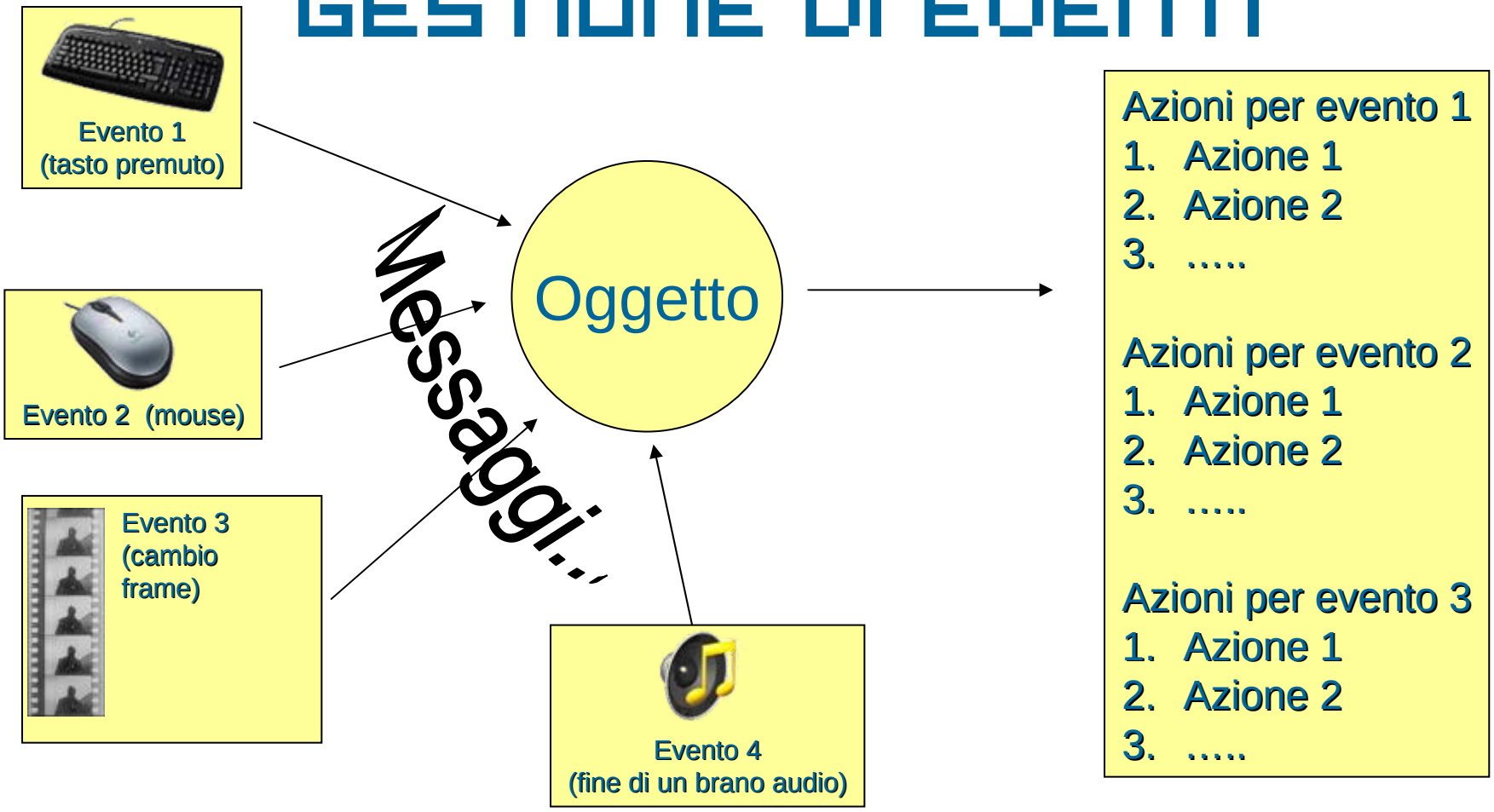
```
function somma(a:Number, b:Number, c:Number):Number {
     return (a + b + c);
}
// sommo tre numeri
trace(somma(1, 4, 6)); // 11
// La somma non è un numero (c è uguale a undefined)
trace(somma(1, 4)); // NaN
// il parametro non dichiarato è ignorato
trace(somma(1, 4, 6, 8)); // 11
```

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.

FUNZIONI ED EVENTI

PROGRAMMAZIONE È GESTIONE DI EVENTI



GESTIONE DI EVENTI IN ACTIONSCRIPT

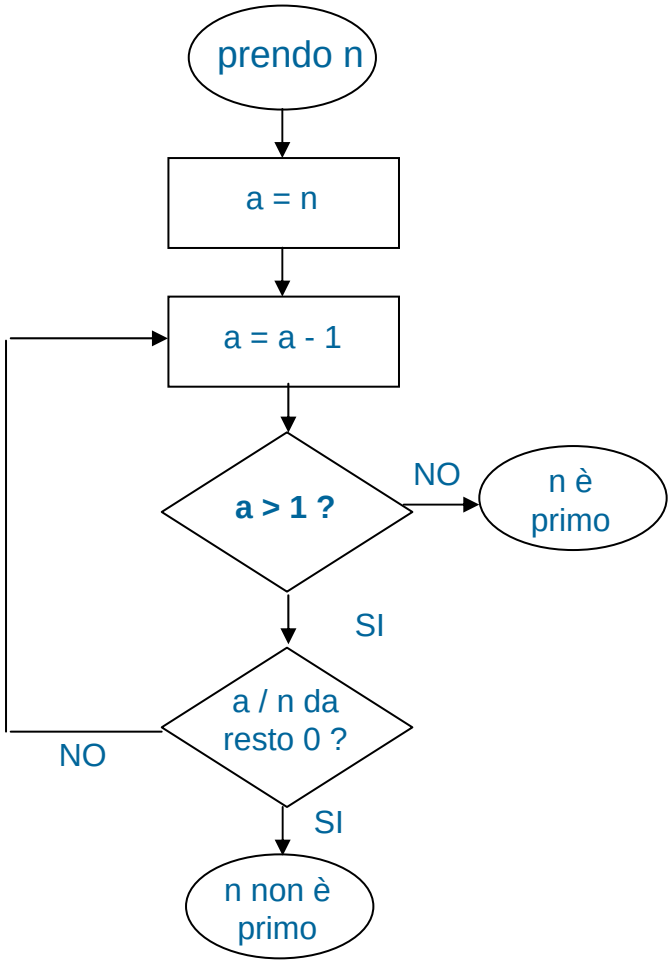
```
//uso di addEventListener
oggetto.addEventListener(nomeEvento:String,
  nomeFunzione:Function);

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

PROGETTAZIONE DI UNA FUNZIONE

NUMERO PRIMO

1: n è il numero da verificare
2: a = n
3: decremento a
4: a è maggiore di 1 ?
5: se no n è primo -> fine
6: a è un divisore di n ?
7: se sì n NON è primo -> fine
8: vado al punto 3



LA FUNZIONE PRIMO IN ACTIONSCRIPT

```
//dichiaro una funzione che accetta come parametro
//un numero e restituisce un valore Boolean (vero o falso)
function primo(n:Number):Boolean {
    /*dichiaro la variabile a di tipo number e le assegno come valore iniziale il valore di n (il numero
    che devo verificare) diminuito di 1*/
    var a:Number = n - 1;
    //inizia un ciclo che continuerà fino a che a è maggiore di 1
    while (a > 1) {
        // se a è un divisore esatto di n (cioè il resto della divisione n/a è 0)...
        if ((n % a) == 0) {
            // n NON è un numero primo per cui restituisco il valore false
            // la funzione termina qui e non viene eseguito alcun altro comando
            return false;
        }
        //altrimenti (se cioè la funzione non si interrompe per effetto
        //dell'istruzione return) diminuisco a di 1 e il ciclo viene ripetuto
        a--;
    }
    // se il ciclo si concluso (a è diventato 1 per diminuzioni successive)
    // significa che non esistono divisori di n e quindi n è primo
    return true;
}
```