

LA LEGGIBILITÀ DEL CODICE

Leggibilità

- Scrivere programmi *sensati* e *leggibili* è difficile, ma molto importante
- È essenziale per lavorare in gruppo
- Aiuto il debugging
- Aiuta a riutilizzare il codice e quindi ci risparmia fatica

Leggibilità significa:

- Progettare con chiarezza
- Scrivere codice con chiarezza

Progettare con chiarezza

- Dedicare il tempo necessario alla progettazione della nostra applicazione non è tempo perso.
- Ci aiuterà a chiarire la logica e la sintassi del nostro lavoro.
- Più avremo sviluppato l'algoritmo che sta alla base della nostra applicazione più il nostro programma sarà comprensibile

Scrivere con chiarezza

- La chiarezza della scrittura si ottiene attraverso due *tecniche* :
- L'***indentazione***: inserire spazi o tabulazioni per mettere subito in evidenza le gerarchie sintattiche del codice.
- I ***commenti***: inserire note e spiegazione nel corpo del codice.

Identazione: un esempio

- Prendiamo in esame questo brano di codice HTML :

```
<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>
```

Identazione: un esempio

- E confrontiamolo con questo:

```
<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>
```

Identazione

- Si tratta della stessa tabella, ma nel primo caso ci risulta molto difficile capire come è organizzata. Nel secondo la gerarchia degli elementi risulta molto più chiara.

Identazione

- L'identazione non ha nessun effetto sulla compilazione del programma
- Serve solo a rendere il nostro lavoro più leggibile.

Inserire commenti

- Rende il codice leggibile anche ad altri
- Quando decidiamo di apportare modifiche a cose che abbiamo scritto ci rende la vita più facile.

Delimitatori

- Delimitatori di riga: tutto ciò 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.

/* ... */ <!-- ... -->

INTRODUZIONE ALLA LOGICA

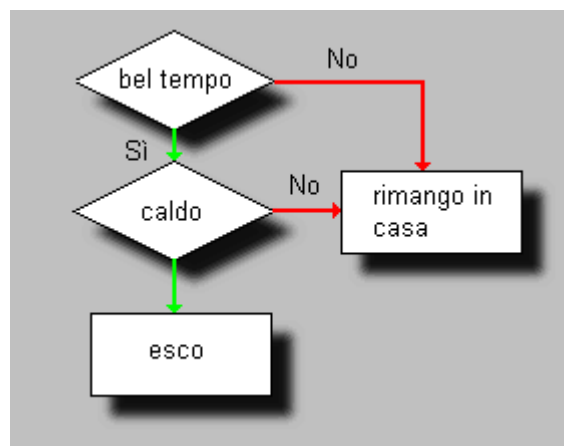
Introduzione

- Nella lezione precedente abbiamo visto che qualsiasi processo logico può essere ricondotto ad una sequenza di eventi elementari (**algoritmo**)
- Che 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).

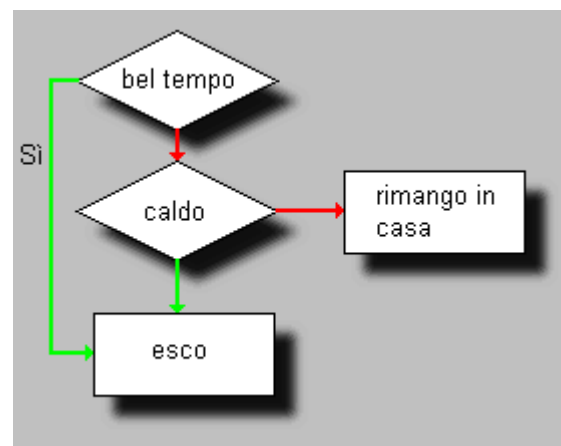
Problema

- Prendiamo questi due enunciati:
 - esco se è bel tempo ed è caldo
 - esco se è bel tempo o se è caldo

Diagrammi di flusso



esco se è bel tempo ed è caldo



esco se è bel tempo o è caldo

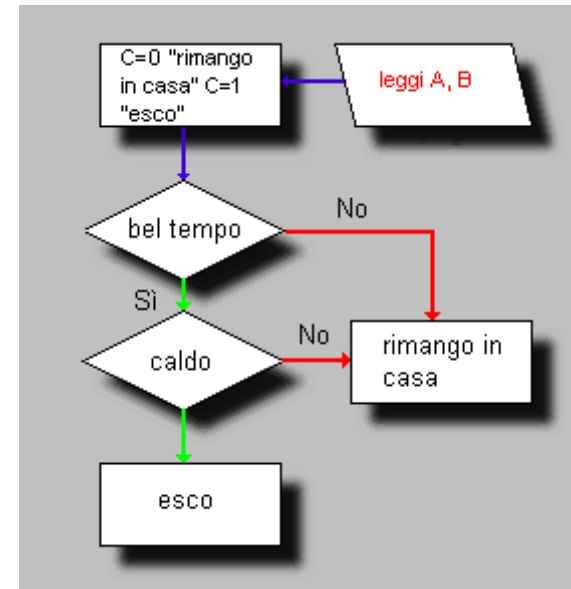
Gli operatori logici

- Come secondo passo, si tratta di convertire i diagramma di flusso in un linguaggio comprensibile dall'elaboratore. Ciò si ottiene con i cosiddetti operatori logici elementari.

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

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

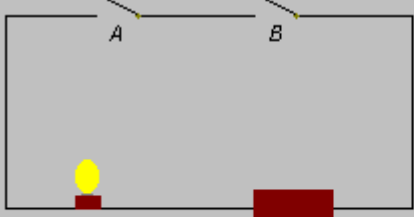
NOT falso	risultato vero
NOT vero	risultato falso

OR - Disgiunzione

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

Gli operatori logici

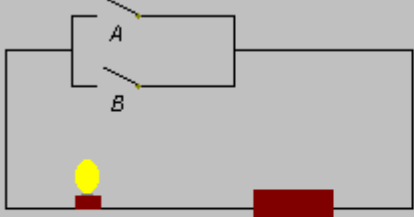
rappresentazione con i circuiti elettrici



se gli interruttori A e B sono entrambi abbassati, il circuito è chiuso e la lampadina si accende

A	B	C
0	0	0
0	1	0
1	1	1
1	0	0

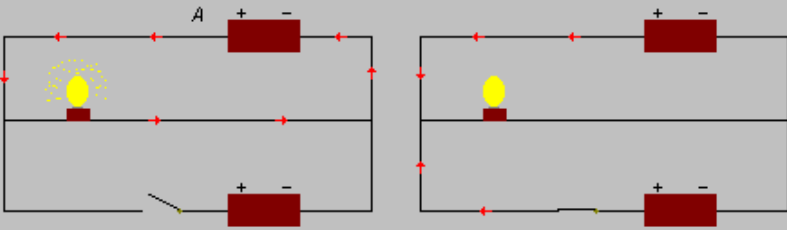
tavola di veridicità per AND



se l'interruttore A o B è abbassato, il circuito è chiuso e la lampadina si accende.

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

tavola di veridicità per OR



se l'interruttore è aperto, la batteria A alimenta il circuito e la lampadina è accesa

se l'interruttore è abbassato, entrambe le batterie alimentano il circuito e la lampadina è spenta

A	C
1	0
0	1

tavola di veridicità per NOT

GLI ELEMENTI DEL LINGUAGGIO

Introduzione

- Nella lezione precedente abbiamo visto che nell'esecuzione di un **algoritmo** entrano in gioco tre tipi di grandezze:
 - Costanti
 - Variabili
 - Espressioni
- In più abbiamo visto che vengono definite un numero finito di azioni elementari

Introduzione

- *Costante*: quantità nota a priori il cui valore non dipende dai dati di ingresso e non cambia durante l'esecuzione (es: valore 0 nell'algoritmo precedente)
- *Variabile*: nome simbolico cui è assegnato un valore che può dipendere dai dati di ingresso e variare durante l'esecuzione (es: A e B)
- *Espressione*: sequenza di variabili, costanti o espressioni combinate tra loro mediante operatori (es: $a+b*4$)

Sintassi

- Ora prenderemo in esame questi elementi in termini grammaticali.
- 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.

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

- Le parole chiave sono i termini (composti da caratteri alfanumerici), riservati 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.
- L'uso improprio di tali termini viene generalmente rilevato durante la fase di compilazione di un programma.

Parole chiave in 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

Parole chiave in 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

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**:

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

Proprietà degli operatori

- **Posizione**

Un operatore si dice **prefisso** se viene posto prima degli operandi, **postfisso** se viene posto dopo e **infisso** se si trova tra gli operandi.

- **Arietà**

L'arietà è il numero di argomenti di un operatore: 1, 2 o 3.

Proprietà degli operatori

- **Precedenza (o Priorità)**

Indica l'ordine con il quale verranno eseguite le operazioni. Ad esempio in **$4+7*5$** verrà prima eseguita la moltiplicazione poi l'addizione.

- **Associtività**

Un operatore può essere associativo a **sinistra** oppure associativo a **destra**. Indica quale operazione viene fatta prima a parità di priorità.

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:

() { } , ;

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, stringe di caratteri, valori di verità (true, false), array, oggetti, ecc.

Costanti numeriche

- Le **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 segnerà 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 segnalare al compilatore che un numero non è decimale si fa precedere il numero da un prefisso. Per i numeri esadecimali questo prefisso è **0x**.
- Gli altri *termini* (*parole chiave* e *nomi*) NON possono iniziare con un numero.

Esempi di costanti numeriche

1

2433

1000000000

3.14

.333333333333

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.
- 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.

Principali sequenze di escape

- `\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).

Esempi di costanti stringa

```
// 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"
```

Costanti booleane

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

Costanti di tipo Array

- 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" ] ;
```

Costanti di tipo Object

- Il letterale ***Object*** è invece compreso tra parentesi graffe ed è costituito da una serie di coppie “**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**.

Identificatori (o Nomi)

- 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).

Regole per gli 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 nome 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

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.

Dichiarazioni di variabili

- Per dichiarare una variabile in ActionScript 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) vengono 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 (inizializzazione; condizione; modifica)
 blocco istruzioni;`
- Il codice <blocco istruzioni> viene eseguito fino a che l'espressione <condizione> risulta vera, poi si passa alla istruzione successiva al **for**.

for e while

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

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

for .. in

- Un ciclo **for..in** consente di eseguire iterazioni scorrendo gli elementi di un oggetto (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

```
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
scrivi sullo schermo "Inserisci un numero maggiore di 2";
intero A = - numero inserito dall'utente -
if (A maggiore o uguale 2) {
    elenco = "0, 1, ";        //se il numero inserito è maggiore di due elenco conterrà
                                //all'inizio i primi due numeri della serie
}
for (intero i = 0; i minore di (A-2); incrementa 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 una virgola e uno spazio
}
// 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;
```

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
```