

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

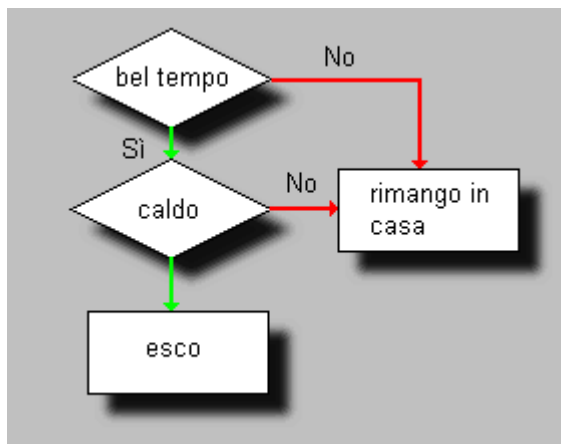
Introduzione

- 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).

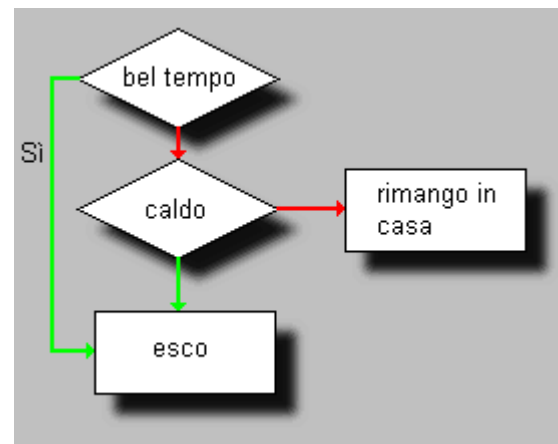
Problema

- 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

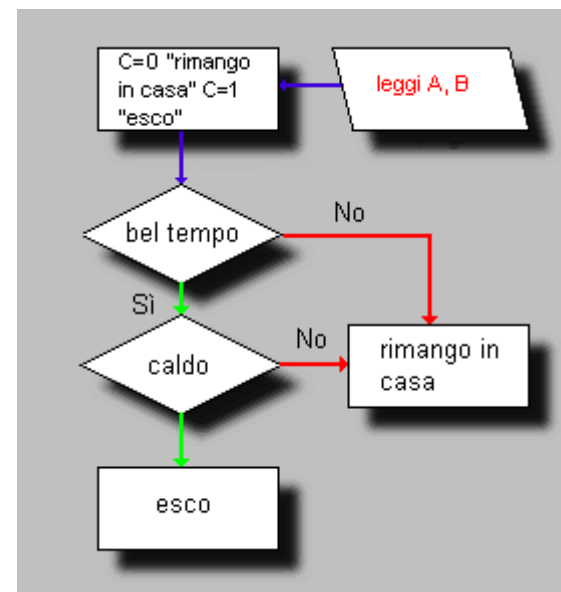
Gli operatori logici

- 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	se (if)	
azione	allora {esecuzione azione}	
coniunzione	e (and &&)	AND
separazione	o (or)	OR
negazione	non (not !)	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 -> 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

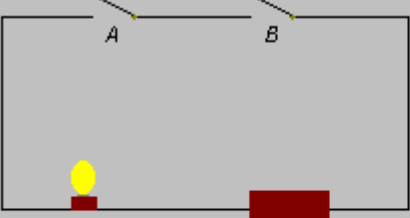
NOT falso	risultato vero
NOT vero	risultato falso

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

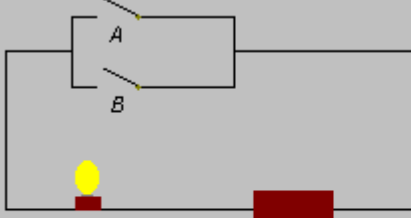
rappresentazione con i circuiti elettrici



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

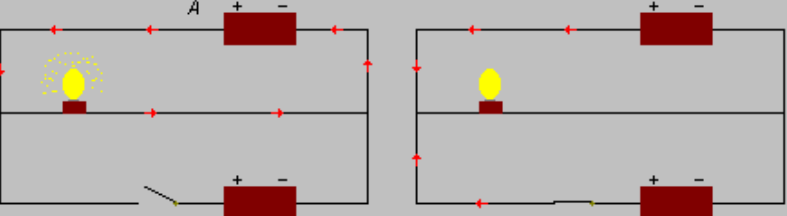
tavola di veridicità per AND



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

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;  
}
```

FUNZIONI E METODI

COSA È UNA FUNZIONE

- Una funzione (o procedura o metodo) è un 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 compongono 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 funzione 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 nomefunzione (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  
}
```

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

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 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.