GLI ELEMENTI DEL LINGUAGGIO

INTRODUZIONE

- **Costante**: quantità nota a priori che non dipende dall'input dell'utente e non cambia durante l'esecuzione del programma.
- **Variabile**: nome simbolico a cui è associato un valore che può dipendere dall'input dell'utente e cambiare durante l'esecuzione del programma.
- **Espressione**: sequenza di variabili, costanti, espressioni collegate tra loro da operatori.

Sintassi

- Ora prenderemo in esame questi elementi in termini grammaticali.
- Come punto di riferimento prenderemo il linguaggi con cui avremo a che fare in questo corso: ***ActionScript*** il linguaggio utilizzato da FLASH, 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

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

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

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

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 segnalerà 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
.33333333333
0.5
2345.675
0xFF0088
0x5500ff
0xff.00aa
```

Costanti stringa

- Una stringa è una sequenza di caratteri **UTF 16** 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.

DATI ELABORATI DA UN PROGRAMMA

PROGRAMMA 2

evento	azioni
Creazione del filmato	1. Crea un oggetto di tipo testo input e assegnagli il nome **numero_txt** 2. Imposta le proprietà visive di **numero_txt** 3. Crea un oggetto di tipo testo dinamico e assegnagli il nome **messaggio_txt** 4. Imposta le proprietà visive di **messaggio_txt**
Primo frame *(il codice viene eseguito ogni volta che viene visualizzato il primo frame)*	1. Dichiara le variabili a e b come **int** (conterranno numeri interi) 2. Prendi il testo presente nella proprietà **text** di numero_txt convertilo in un numero intero e metti il risultato nella varibile **a**. 3. Calcola il quadrato di **a** e metti il risultato in **b**. 4. Converti a e b in testo (string), componi il messaggio che comunica il risultato e modifica la proprietà text di messaggio_txt in modo che mostri il messaggio.

PROGRAMMA 2

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

PROGRAMMA 2

`a`	variabile	`int`
`b`	variabile	`int`
`"Il quadrato di "`	costante	`string`
`" è "`	costante	`string`
`a*a`	espressione	`int`
`"Il quadrato di " + a.toString() + " è " + b.toString();`	espressione	`string`

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.

USO DELLE 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 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

evento	azioni
Creazione del filmato	1. Crea un oggetto di tipo testo input e assegnagli il nome **numero_txt** 2. Imposta le proprietà visive di **numero_txt** 3. Crea un oggetto di tipo testo dinamico e assegnagli il nome **messaggio_txt** 4. Imposta le proprietà visive di **messaggio_txt**
Primo frame *(il codice viene eseguito ogni volta che viene visualizzato il primo frame)*	1. Dichiara le variabili a e b come **int** (conterranno numeri interi) 2. Prendi il testo presente nella proprietà **text** di numero_txt convertilo in un numero intero e metti il risultato nella varibile **a**. 3. Calcola il quadrato di **a** e metti il risultato in **b**. 4. Converti a e b in testo (string), componi il messaggio che comunica il risultato e modifica la proprietà text di messaggio_txt in modo che mostri il messaggio.

PROGRAMMA 2

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

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

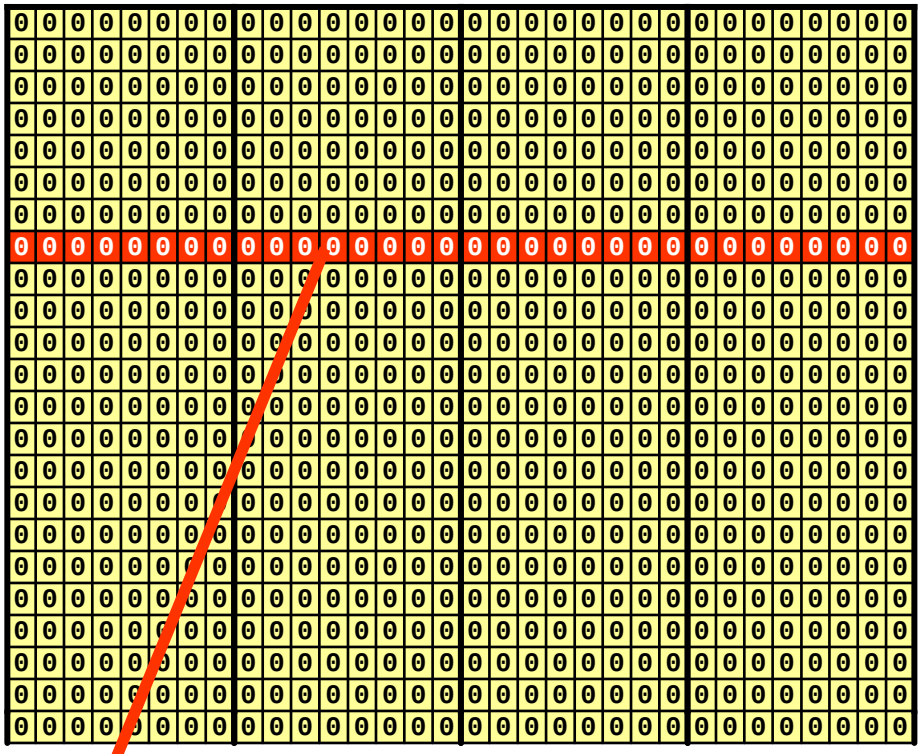
- Quando io dichiaro una variabile il compilatore riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un indirizzo anch'esso espresso in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

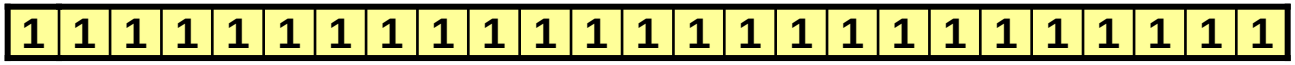


```
var a:int;
```

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;
```



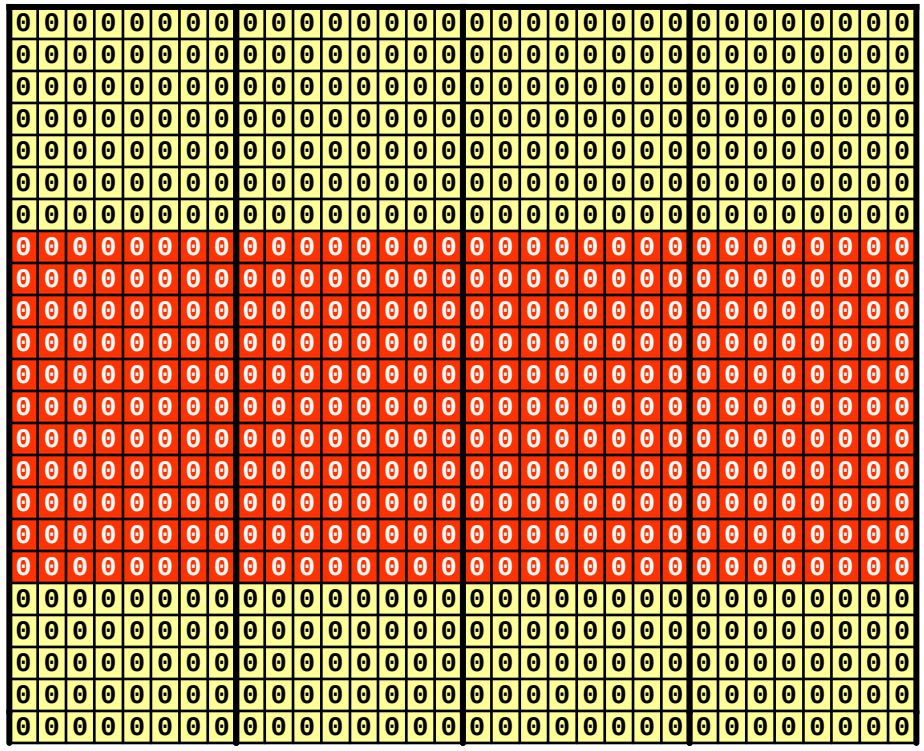
VALORI E PUNTATORI

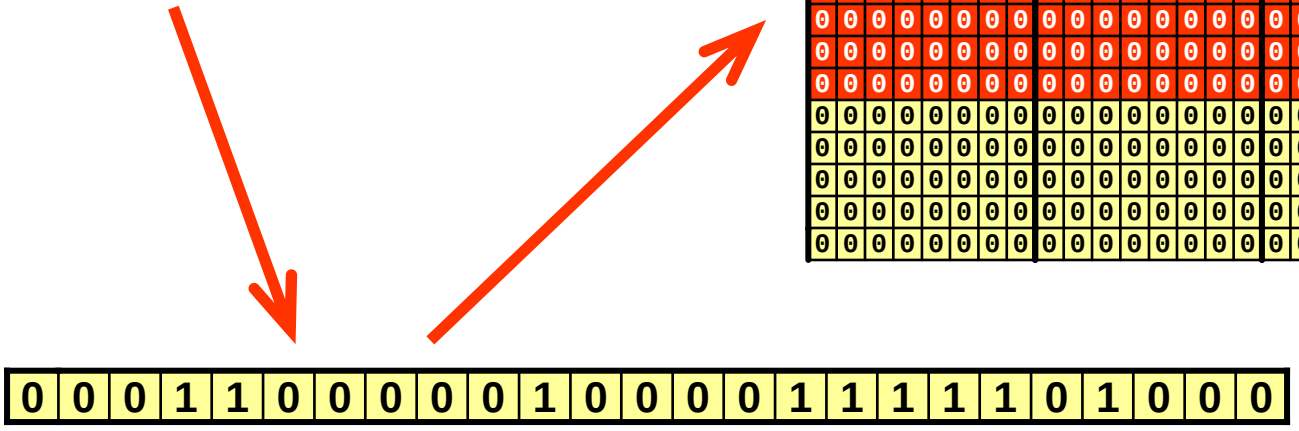
- Quando la casella che la variabile rappresenta contiene direttamente il dato si dice che la varibile **contiene un valore**.

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.

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 esamo 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 delle 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 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.

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