ALGORITMI

FUNZIONAMENTO DEL CALCOLATORE

- Un computer per poter risolvere un qualsiasi problema ha bisogno di qualcosa che gli indichi esattamente cosa fare passo dopo passo
- Un computer si limita ad eseguire delle istruzioni "*macchina*" per l'esecuzione delle quali e' stato progettato
- Un'istruzione "*macchina*" e' un'istruzione "primitiva" comprensibile e direttamente eseguibile dal calcolatore senza bisogno di essere interpretata

SOFTWARE

- Qualsiasi compito per poter essere eseguito da un calcolatore deve essere tradotto in un insieme di istruzioni *macchina*
- Un software non e' altro che un insieme di istruzioni eseguibili dal calcolatore che nel loro insieme risolvono un problema o eseguono un determinato compito
- E' compito del programmatore "tradurre" un problema utilizzando solamente il set di istruzioni "primitive" comprensibili al calcolatore

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.

PROBLEMA

Problema:

Un contadino deve fare attraversare il fiume ad una capra, un cavolo, e un lupo, avendo a disposizione una barca in cui può trasportare solo uno dei tre alla volta. Se incustoditi, la capra mangerebbe il cavolo e il lupo mangerebbe la capra

SOLUZIONE

Algoritmo che descrive la soluzione:

1. Porta la capra sull'altra sponda
2. Porta il cavolo sull'altra sponda
3. Riporta la capra sulla sponda di partenza
4. Porta il lupo sull'altra sponda
5. Porta la capra sull'altra sponda

PROPRIETÀ FONDAMENTALI DEGLI ALGORITMI

- *Non-ambiguita'*: il procedimento deve essere interpretabile in modo univoco da chi lo deve eseguire (l'ordine di esecuzione dei passi deve essere non ambiguo)
- Eseguibilita': ogni istruzione dell'algoritmo deve poter essere eseguita senza ambiguità da un esecutore reale o ideale
- Finitezza: sia il numero di istruzioni che compongono l'algoritmo che il tempo di esecuzione dello stesso devono essere finiti

ALGORITMO BASE DEI CALCOLATORI

Finché non trovi un'istruzione di *halt* fai:

- Leggi un'istruzione dalla memoria
- Decodifica l'istruzione appena letta
- Esegui l'istruzione

RAPPRESENTAZIONE DI UN ALGORITMO

- Rappresentazione mediante **pseudolinguaggio** o **pseudocodice** che possiamo definire come un *linguaggio informale (ma non ambiguo) per la descrizione delle istruzioni.* Ogni riga rappresenta un'istruzione. Se non diversamente specificato, le righe si leggono dall' alto verso il basso.
- rappresentazione mediante **diagramma di flusso** (flowchart) che è una rappresentazione grafica in cui ogni istruzione è descritta all'interno di un riquadro e l'ordine di esecuzione delle istruzioni è indicato da frecce di flusso tra i riquadri.

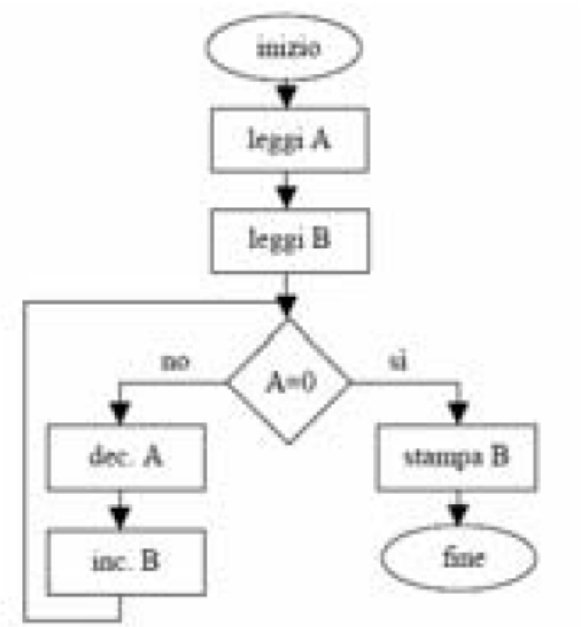
PROBLEMA:

sommare due numeri naturali utilizzando solo incrementi e decrementi

Pseudolinguaggio

1. Leggi A
2. Leggi B
3. Se A=0 vai all'istruzione 7
4. Decrementa A
5. Incrementa B
6. Vai all'istruzione 3
7. Stampa B

Diagramma di flusso



AZIONI PRIMITIVE

- La definizione di funzioni primitive permette di eliminare l'ambiguità

  Es: "Andare a lezione può essere irritante"

  - "Le lezioni causano irritazione"
  - "Il tragitto per arrivare dove si svolgono le lezioni e' irritante"

- Il livello di astrazione delle primitive deve essere scelto in funzione dell'esecutore

- L'insieme delle primitive più un set di regole per metterle insieme costituisce un *Linguaggio di programmazione*

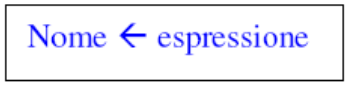
DEFINIZIONE DI UNO PSEUDOLINGUAGGIO

Istruzioni di inizio e di fine	**start**, **end**	Sono i punti d' ingresso e di uscita del flusso di esecuzione
Istruzione di assegnamento	nome←espressione	L' esecuzione di un' istruzione di assegnamento avviene in due fasi: la valutazione dell' espressione (utilizzando i valori correnti delle eventuali variabili che in essa compaiono) e l' aggiornamento del valore della variabile a sinistra del segno
Scelta tra due possibili attività	**if** (condizione) **then** (attività 1) **else** (attività 2)	Biforcazione del flusso in due percorsi alternativi (mutuamente esclusivi) la cui esecuzione è condizionata al verificarsi di una condizione. L' esecuzione di un' istruzione condizionale comporta innanzitutto la valutazione della condizione, e quindi l' esecuzione delle istruzioni che compongono uno dei due cammini.
Strutture iterative	**while** (condizione) **do** (azione)	Esecuzione ripetuta di una o più istruzioni. La ripetizione dipende dall' esito di un controllo. È fondamentale che l' esito del controllo dipenda da variabili il cui valore può essere modificato dall'esecuzione delle istruzioni del ciclo. In caso contrario il ciclo verrebbe eseguito o mai (risultando inutile) o all' infinito (violando la definizione di algoritmo). Le variabili da cui dipende il controllo sono dette variabili di controllo del ciclo. Ogni ciclo è composto da 4 elementi: l' inizializzazione delle variabili di controllo, il controllo della condizione da cui dipende l' iterazione, il corpo del ciclo (sequenza delle istruzioni da eseguire ciclicamente) e l' aggiornamento delle variabili di controllo.
Istruzione di salto	**goto** (riga di destinazione)	L'istruzione di salto sposta il flusso di esecuzione in un particolare punto dell'algoritmo. La riga di destinazione rappresenta, appunto, il punto in cui l' esecuzione riprenderà dopo il goto.
Definizione di procedure	**procedure** nome (istruzione 1 istruzione 2 istruzione 3)	Unità di programma utilizzabili attraverso invocazione da un qualsiasi punto del codice. Tutti i linguaggi di programmazione utilizzano questa strategia per rendere più leggibile il codice. Sinonimi di procedura sono: funzione, metodo, subroutine, sottoproramma ecc

DIAGRAMMA DI FLUSSO

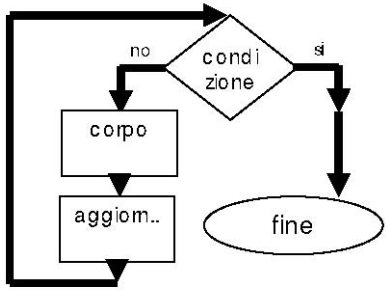
Inizio e fine

Assegnamento ed elaborazione



Scelta condizionata

Iterazione e salti



ESECUZIONE

Ecco come risulta l'agoritmo per sommare due numeri naturali utilizzando solo incrementi e decrementi:

1. **Start**
2. A ← input
3. B ← input
4. **If**(A=0) **then** (
5. **goto**(9))
6. A ← A-1
7. B ← B+1
8. **goto**(4)
9. Stampa B
10. **end**

GRANDEZZE

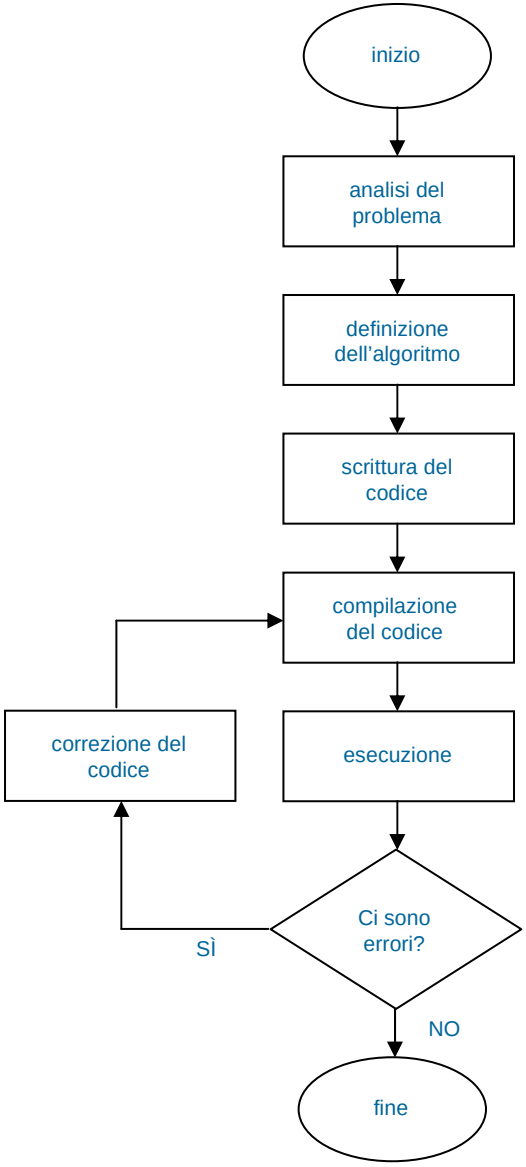
- *Costante*: quantità nota a priori il cui valore non dipende dai dati di ingresso e non cambia durante l'esecuzione (es: valore 0 nell'algoritmo precedente)
- *Variabile*: nome simbolico cui e' assegnato un valore che puo' dipendere dai dati di ingresso e variare durante l'esecuzuione (es: A e B)
-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

COMPILAZIONE

Qualsiasi sia il linguaggio che abbiamo scelto il nostro codice dovrà essere tradotto, perché possa essere eseguito dal computer, in una serie di istruzioni che la CPU è in grado di interpretare ed eseguire. Questo processo, che chiameremo compilazione, a secondo dell'ambiente e del linguaggio scelto potrà avere diverse caratteristiche ma sarà comunque un passaggio necessario del nostro processo creativo.

IL PROCESSO

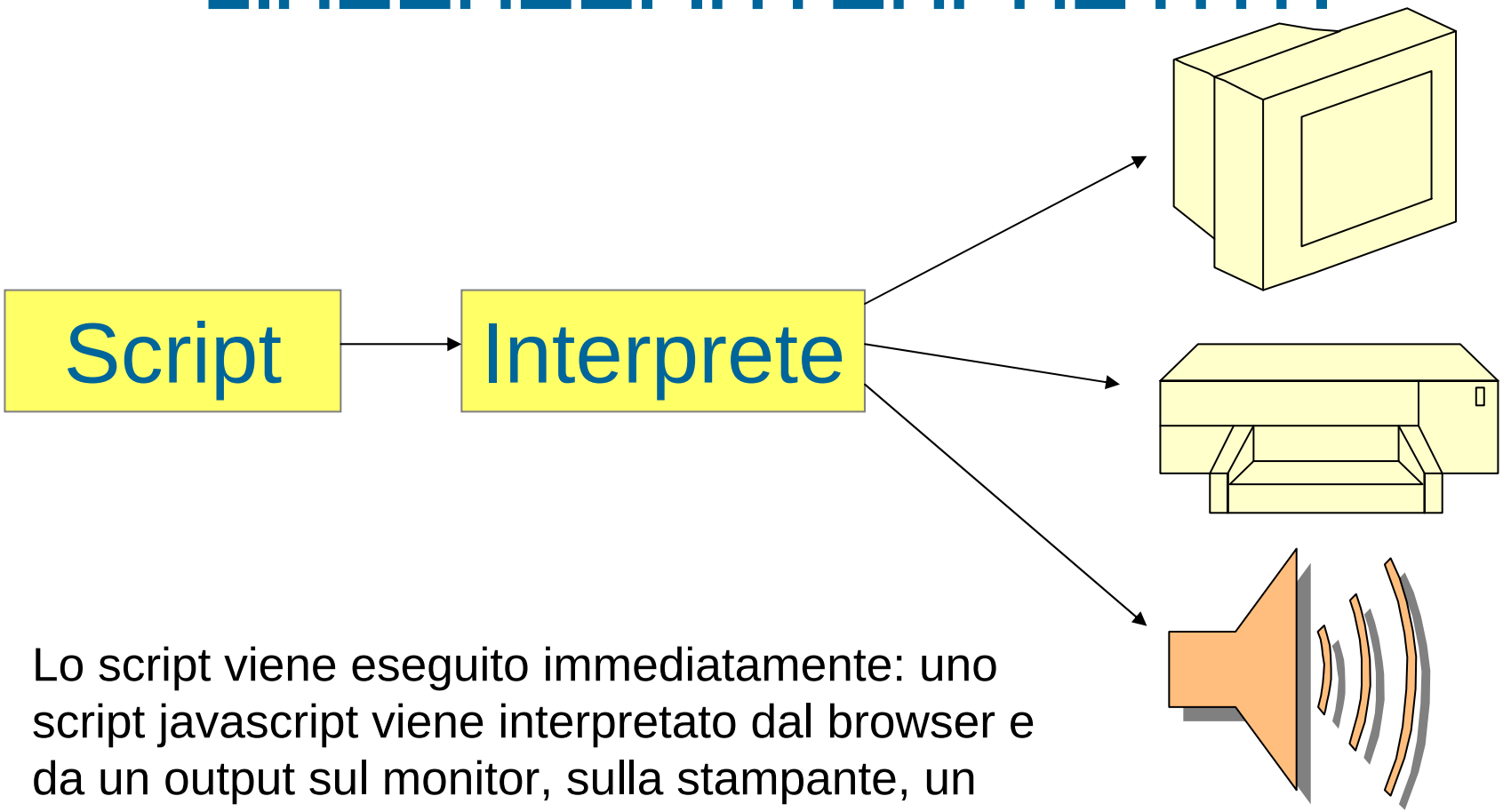
Se utilizziamo un diagramma di flusso per descrivere il processo di creazione di un'applicazione otterremo lo schema a fianco.



LINGUAGGI INTERPRETATI E COMPILATI

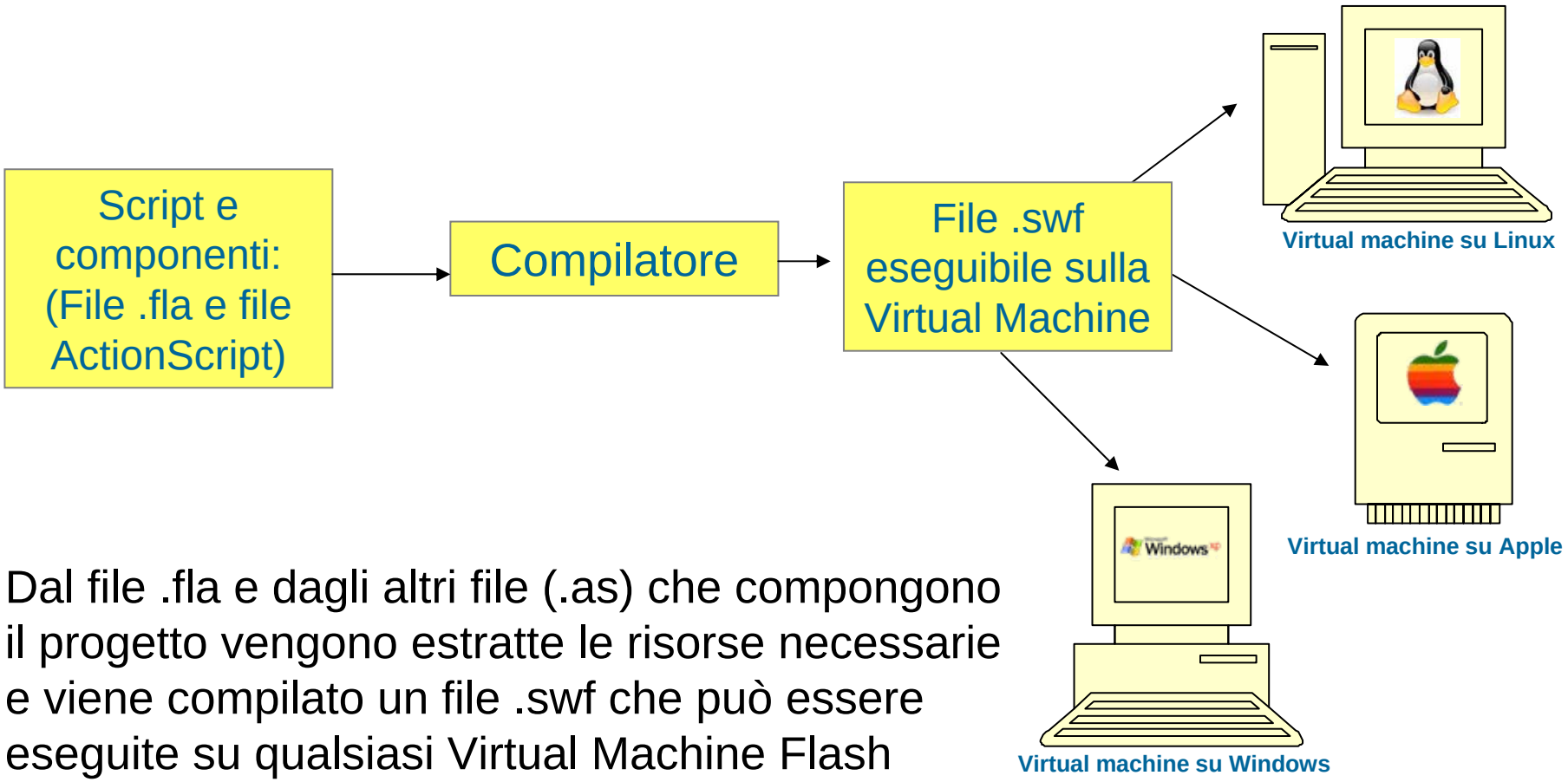
- Per quanto lo sviluppo di applicazioni possiamo distinguere i linguaggi di programmazioni indue grandi categorie;
  - I **linguaggi interpretati**: uno specifico modulo software (detto appunto interprete) esegue direttamente gli script così come li ho composti
  - I **linguaggi compilati**: prima dell'esecuzione uno specifico programma (detto compilatore) combine il codice ed eventualmente altre risorse in un nuovo file che può essere eseguito (o da un sistema operativo specifico o da una Virtual Mchne)

LINGUAGGI INTERPRETATI



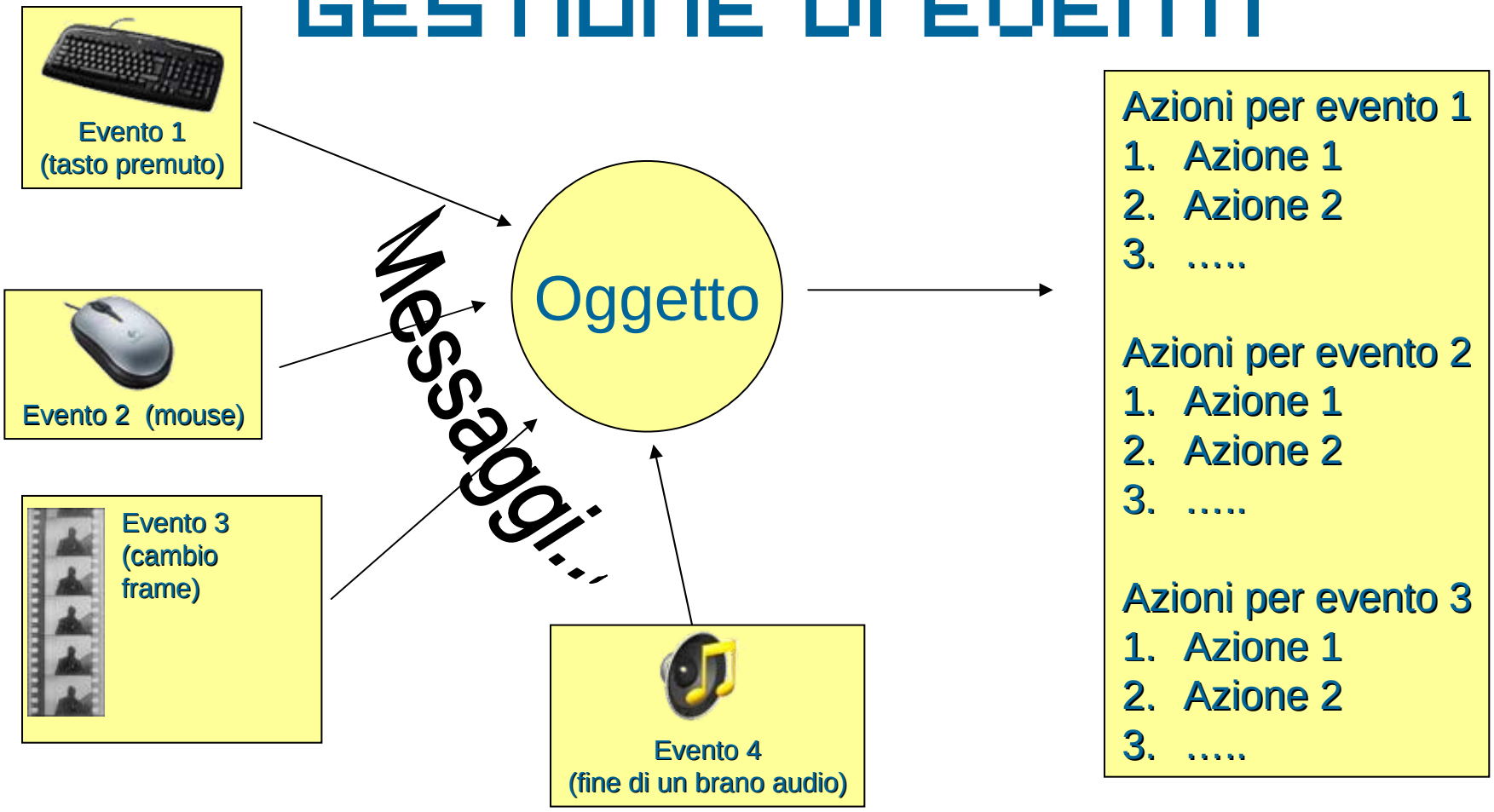
Lo script viene eseguito immediatamente: uno script javascript viene interpretato dal browser e da un output sul monitor, sulla stampante, un output audio, ecc.

LINGUAGGI COMPILATI (ESEMPIO ACTIONSCRIPT)



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

PROGRAMMAZIONE È GESTIONE DI EVENTI



PROGRAMMA 1

```
stop();
prova_txt.text = “Ciao gente”;
```

PROGRAMMA 1

evento	azioni
Creazione del filmato	1. Crea un oggetto di tipo dinamico e assegnagli il nome **prova_txt** 2. Imposta le proprietà di **prova_txt** 3. Posizione **x** = … 4. Posizione **y** = … 5. Carattere = Arial 6. Dimensione Carattere = 20
Primo frame	1. Ferma il filmato su questo frame 2. Modifica la proprita testo di prova_txt: assegna alla proprietà il valore “Ciao gente”.

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

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.

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, 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.