RIPASSO

VARIABILI

- Una variabile è composta da due elementi: il suo **nome** e il suo **valore**; 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

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

a = parseInt(input_txt.text);
b = a*a;

messaggio_txt.text = "Il quadrato di " + a +" è "
   + b;
```

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

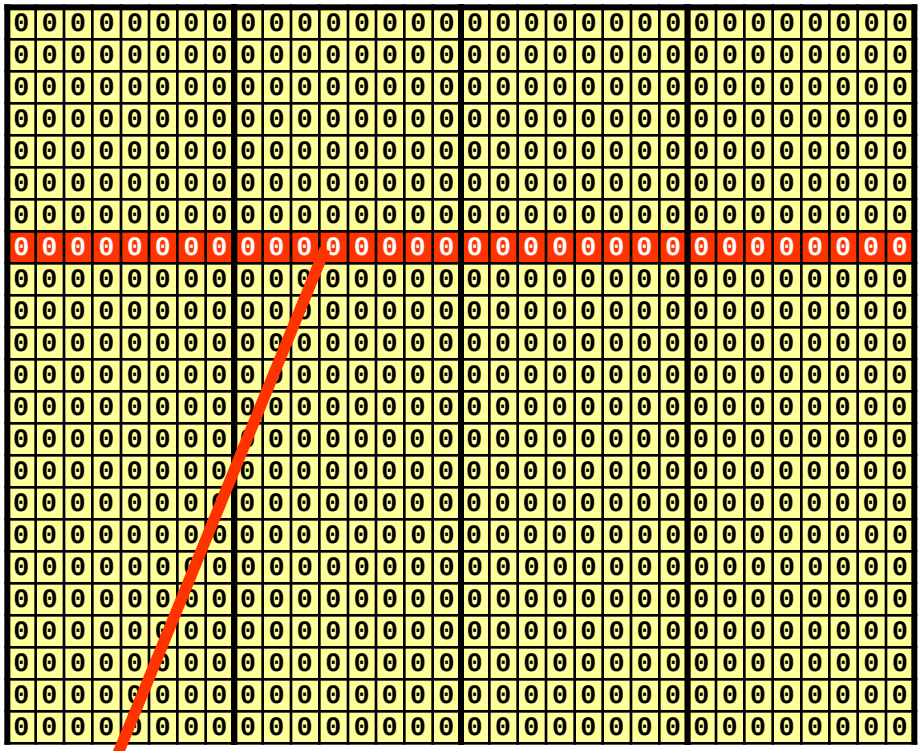
- Quando io dichiaro una variabile il compilatore riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un indirizzo anch'esso espresso in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

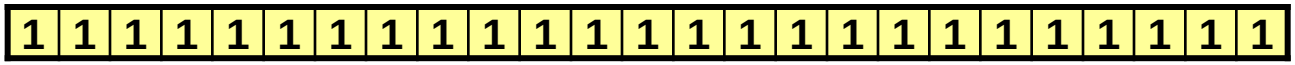


```
var a:int;
```

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;
```



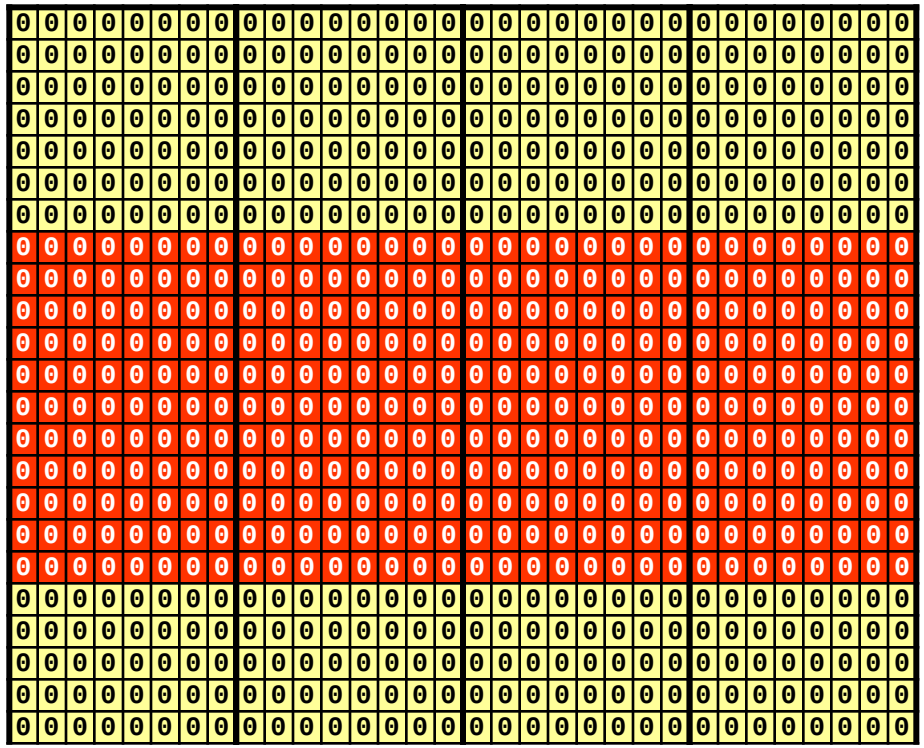
VALORI E PUNTATORI

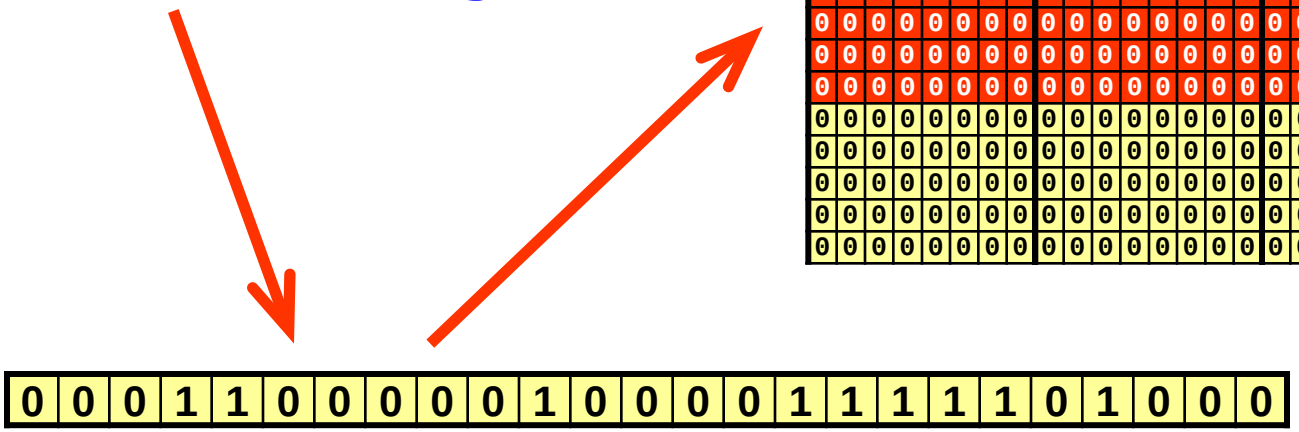
- Quando la casella che la variabile rappresenta contiene direttamente il dato si dice che la varibile contiene un valore.

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**.

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;
```

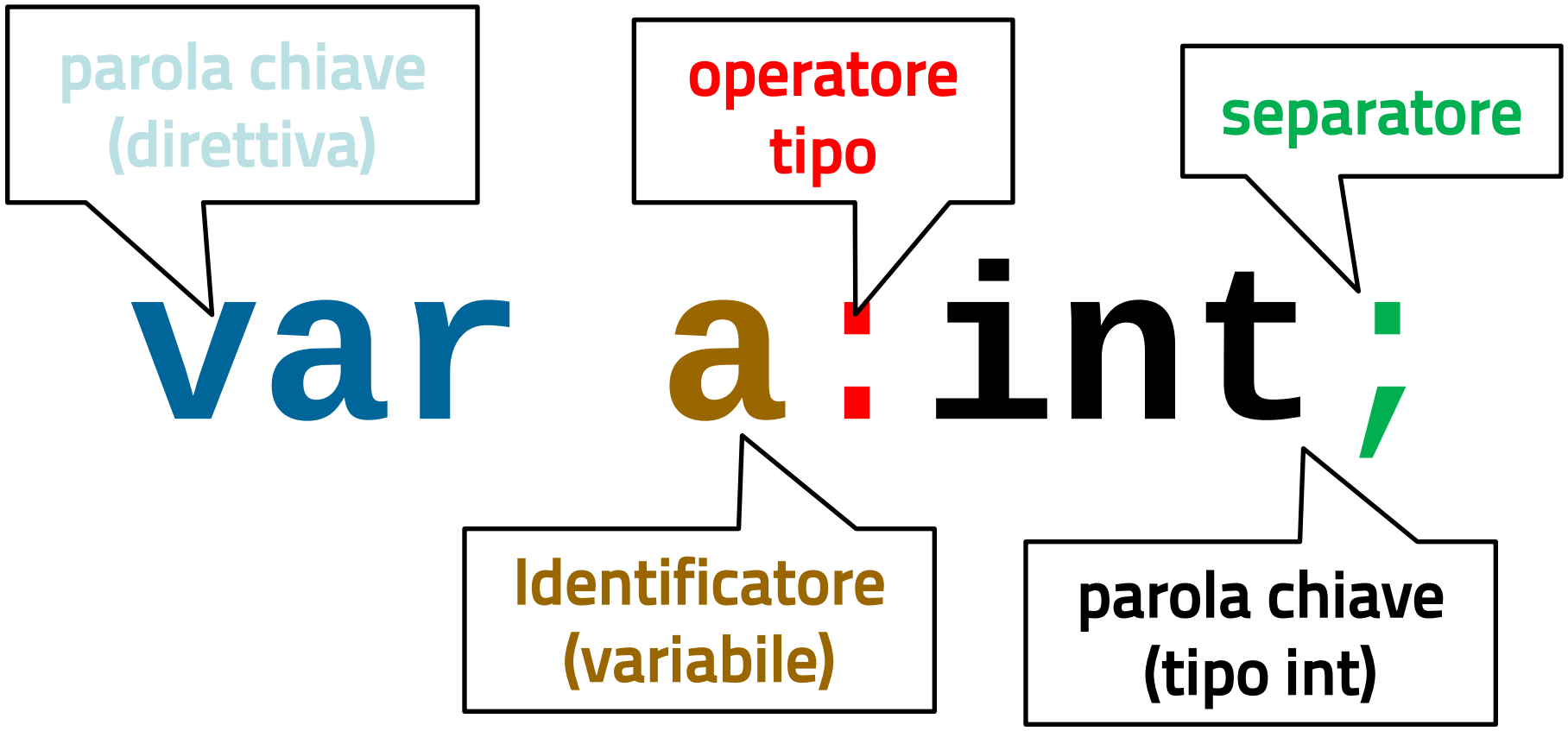
- Opzionalmente si può assegnare un valore alla variabile all'atto della dichiarazione (inizializzazione):

```
var pippo:String = "Hello World" ;
```

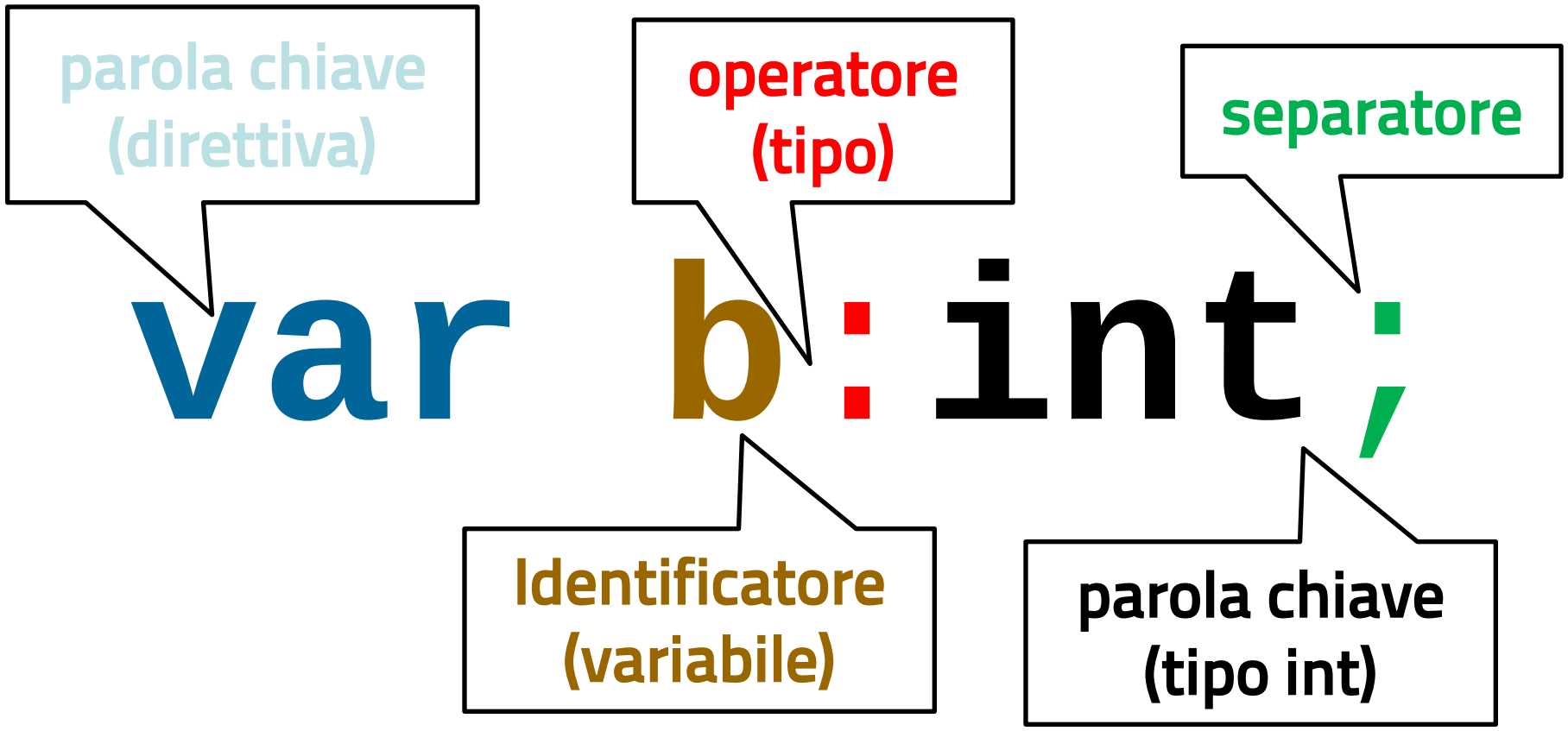
ISTRUZIONI

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

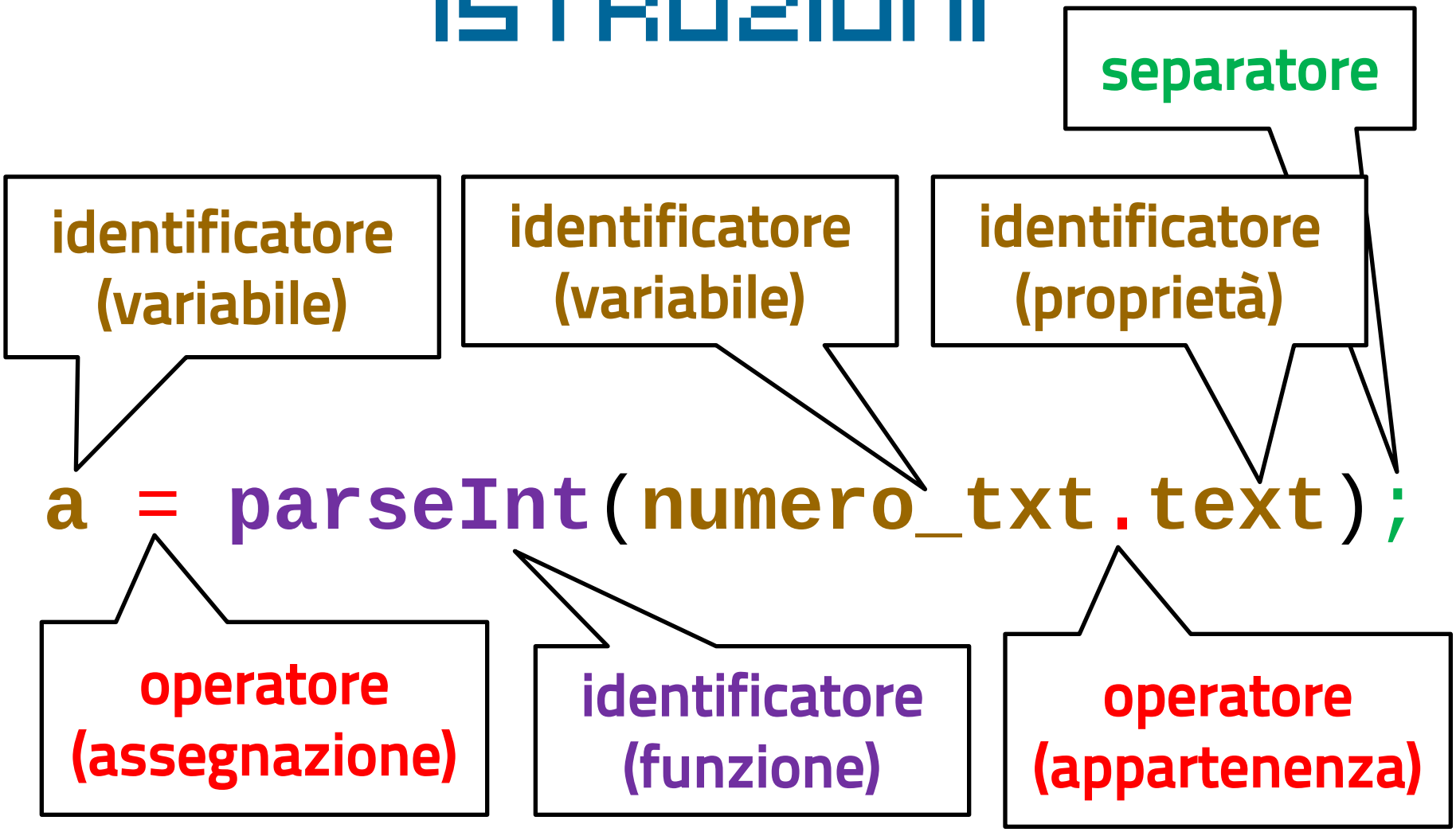
ISTRUZIONI



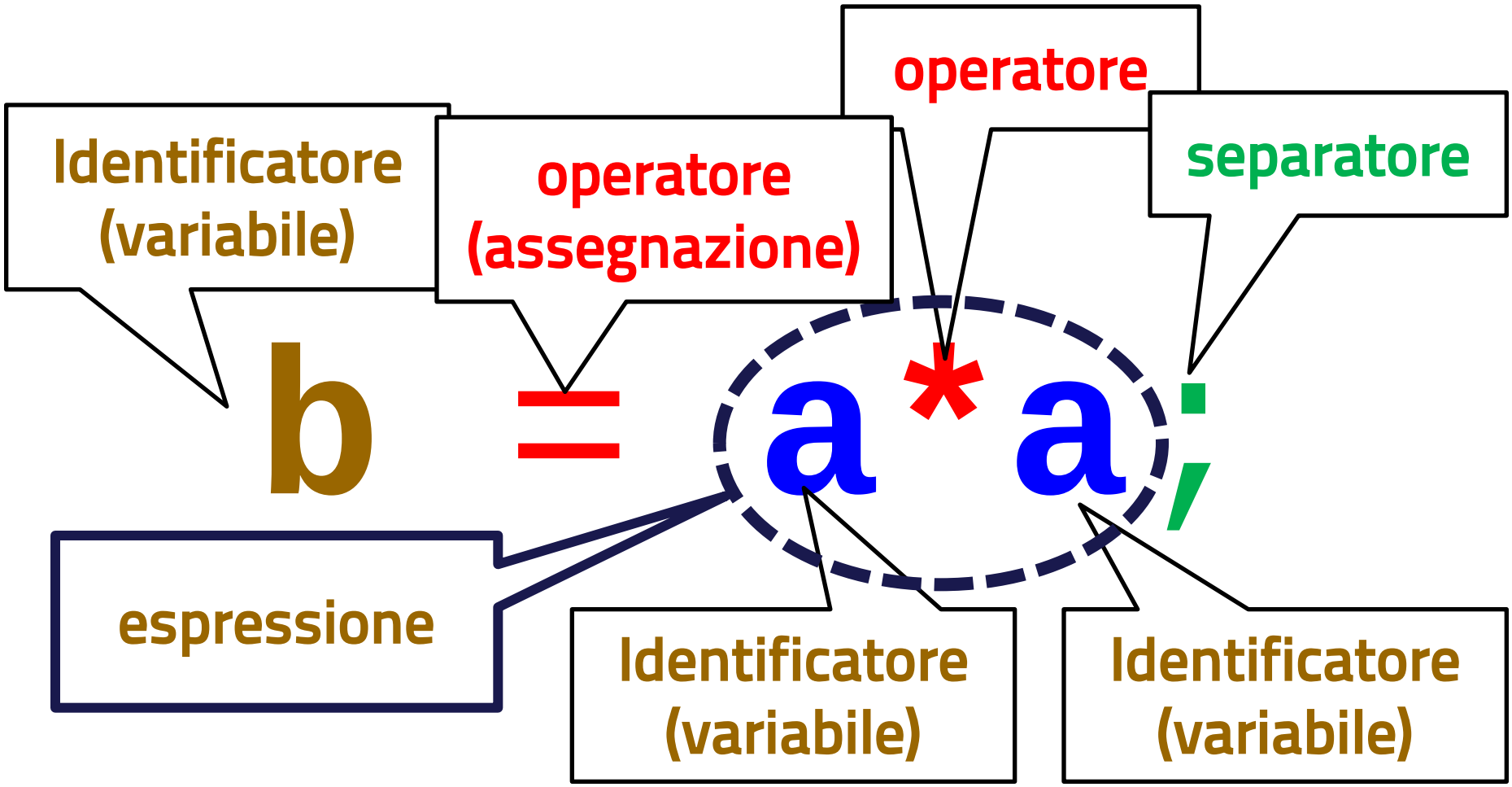
ISTRUZIONI



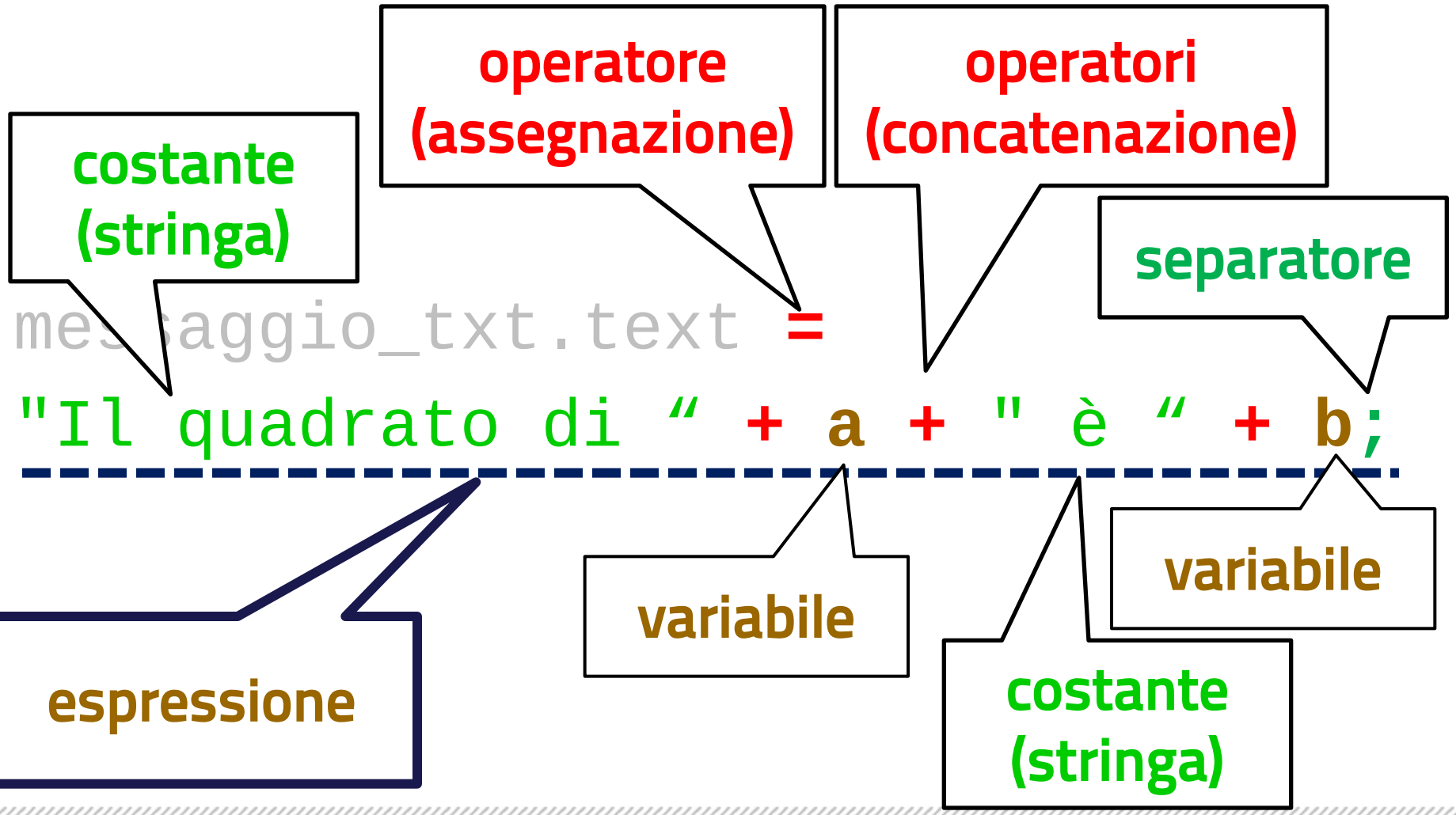
ISTRUZIONI



ISTRUZIONI



ISTRUZIONI



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.

PROGRAMMAZIONE È GESTIONE DI EVENTI

GESTIONE DI EVENTI IN ACTIONSCRIPT

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

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

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

```
var 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

- **Flusso naturale del programma:**
  - 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.

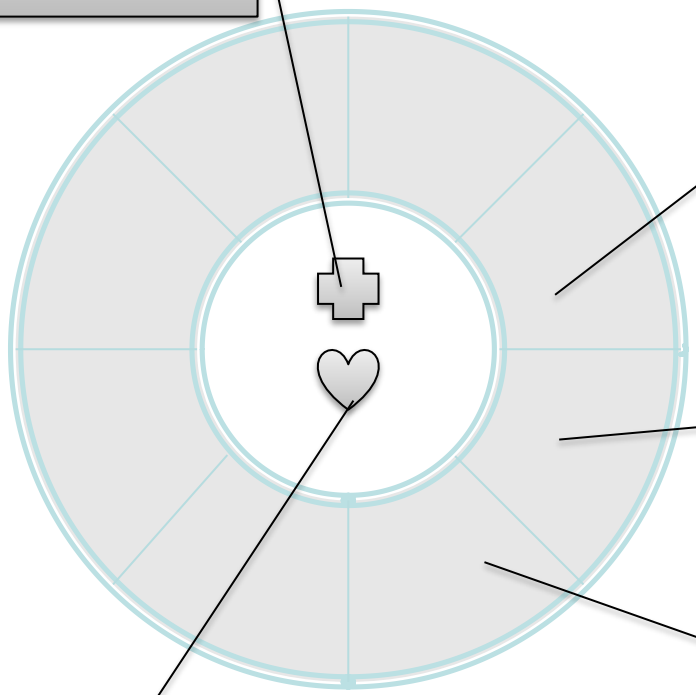
for e while

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

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

LE CLASSI

RUOTA ANIMATA



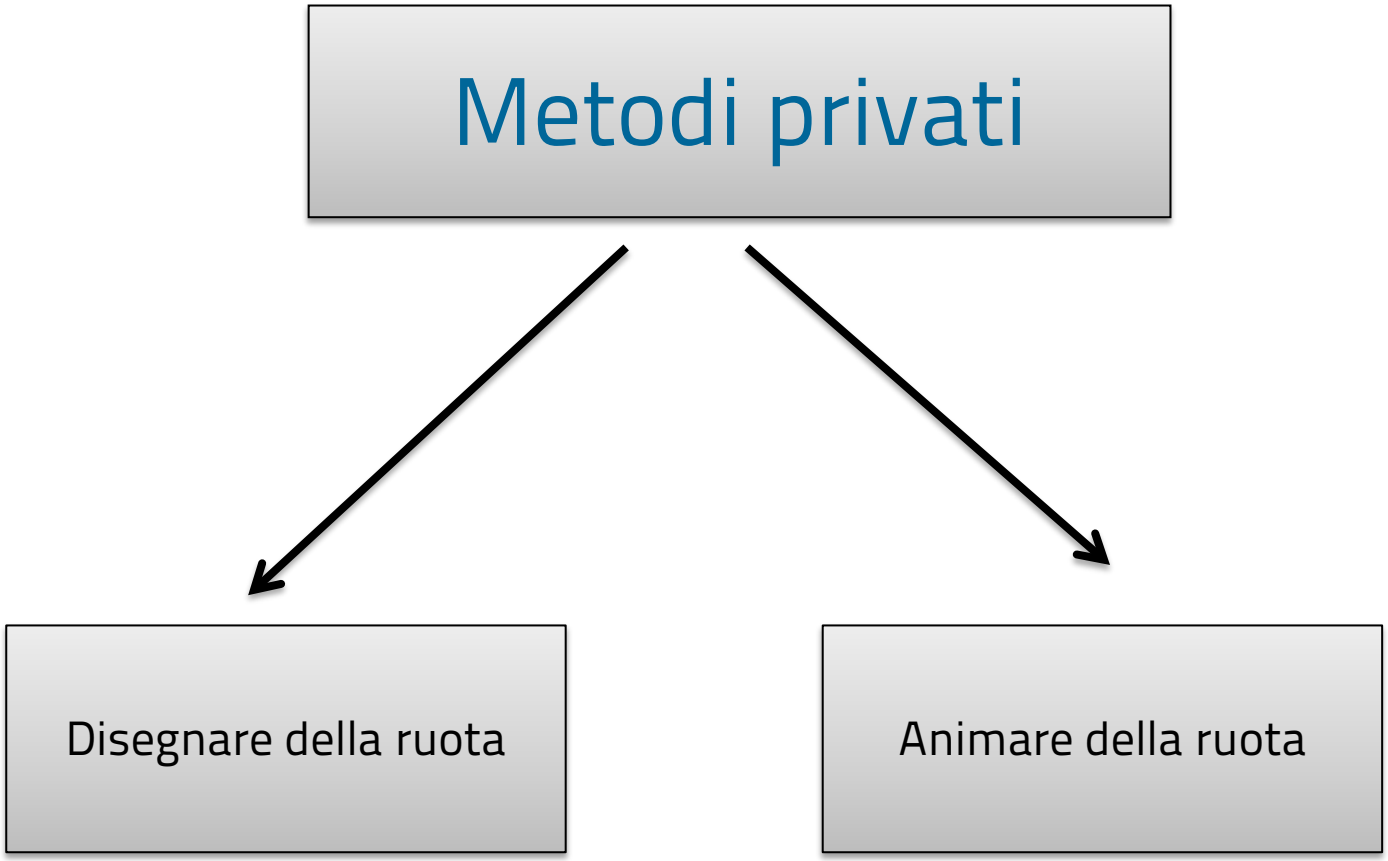
velocità

constructor

setSpeed
(imposta velocità)

getSpeed
(restituisce velocità)

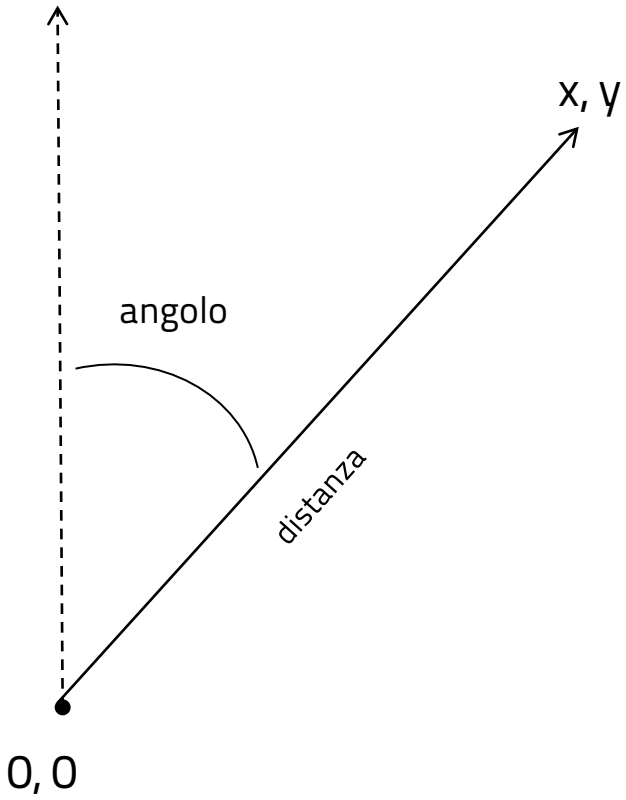
raggio



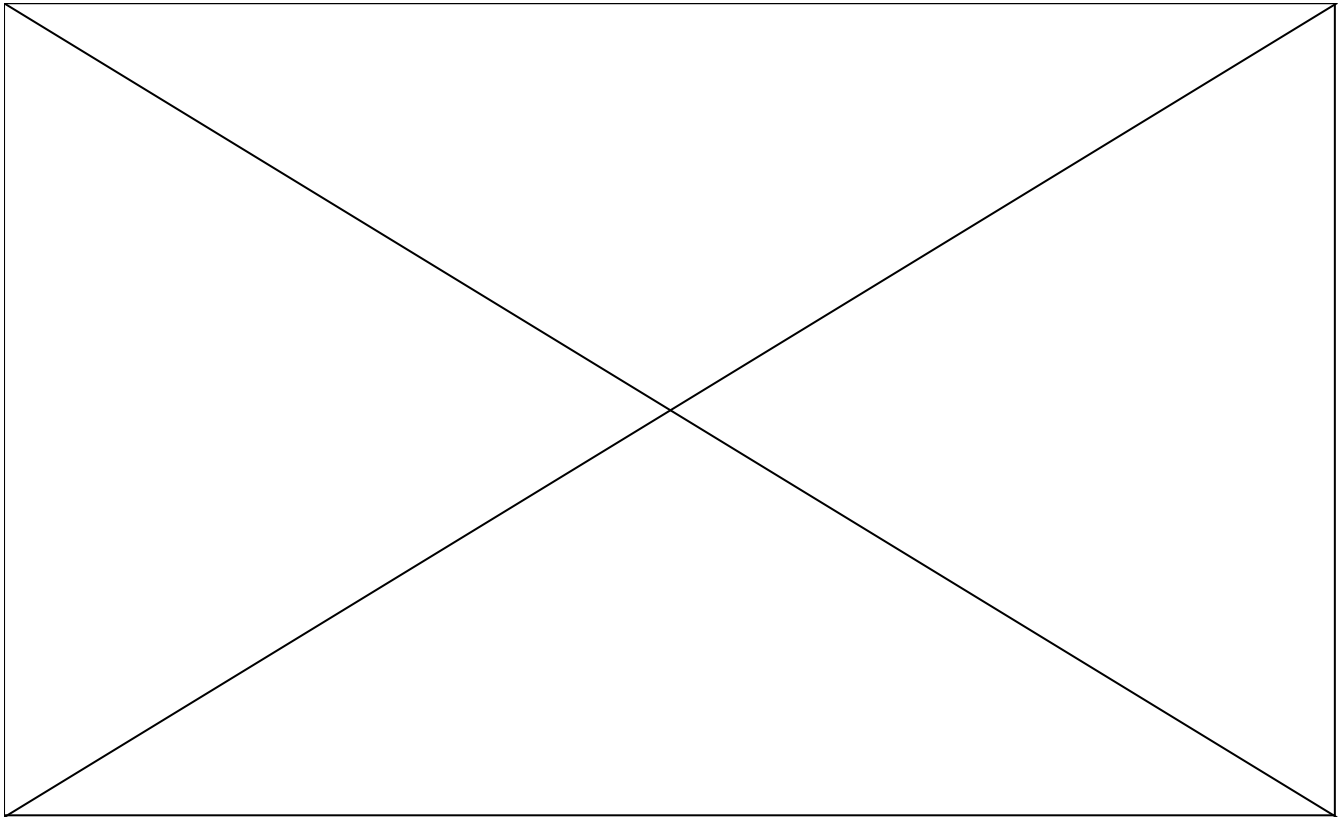
DISEGNARE LA RUOTA

Giro completo = 2*∏;
Un raggio ogni 2*∏/24;
Step = ∏/12;

Point.polar(distanza, angolo);



DISEGNARE LA RUOTA



ANIMAZIONE

1. L'animazione è basata sull'evento ENTER_FRAME
2. Verifico quanti millisecondi sono passati dall'ultime frame
3. Faccio ruotare la ruota in proporzione alla velocità impostata
4. Se la velocità è positiva ruoto in senso orario altrimenti in senso antiorario

PROVARE UNA CLASSE

- Per creare e usare una classe è necessario:
  - Definizione di una classe in un file di classe *ActionScript* esterno.
  - Salvataggio del file di classe nella directory specificata per il percorso della classe (o nel percorso in cui Flash cerca le classi) oppure nella stessa directory del file FLA dell'applicazione.
  - Creazione di un'istanza della classe in un altro script, ossia un documento FLA o un file di script esterno, oppure tramite creazione di una sottoclasse basata sulla classe originale.

USARE UNA CLASSE

- Per creare un'istanza di una classe *ActionScript*, si utilizza l'operatore ***new*** per richiamare la funzione di costruzione della classe. Tale funzione ha sempre lo stesso nome della classe e restituisce un'istanza della classe che generalmente viene assegnata a una variabile.

```
var ruota:RuotaAnimata = new RuotaAnimata();
```

- Usando l'operatore punto (.) si accede al valore di una proprietà o a un metodo di un'istanza.

```
Bici.setSpeed(10);
```

OROLOGIO

INGEGNERIZZARE UN PROBLEMA

MOTORE

(Orologio generico)

Aggiornamento periodico dell'ora ricavandola dall'orologio del computer

VISUALIZZAZIONE

(Orologio digitale)

(Orologio analogico)

VISUALIZZAZIONE

(Orologio digitale)

(Orologio analogico)

- Inizializzazione
- Aggiornamento dell'ora

OROLOGIO GENERICO

- Come costructor definisco una funzione che chiama i metodi necessari a disegnare l'orologio e a inserirvi una valore iniziale:

```
package {
  public class Orologio extends Sprite {
    public function Orologio ():void
    {
// Preparo il video perchè possa mostrare l'ora
         inizializzaVideo();
// Mostro l'ora corrente sul video
        aggiornaOrologio();
// Creo il time che aggiornerà l'orologio una volta la sevondo
        inizializzaTimer();
              }
  .......
}
```

OROLOGIO

- Definisco i metodi che inizializzano il timer e lo fanno partire: leggiOra aggiorna le varibili sulla base dell'ora fornita dal computer:

```
package {
  public class Orologio extends Sprite {
  .......
    protected function leggiOra() {
      var adesso:Date = new Date();
      ore = adesso.getHours();
      minuti = adesso.getMinutes();
      secondi = adesso.getSeconds();
    }
  .......
}
```

OROLOGIO

- inizializzaTimer e aggiorna sono rispettivamente il metodo che crea e fa partire il timer e il metodo che viene chiamato ad ogni evento generato dal timer:

```
package {
  public class OrologioGenerico extends Sprite {
  .......
    protected function inizializzaTimer() {
        var myTimer:Timer = new Timer(1000);
        myTimer.addEventListener(TimerEvent.TIMER, aggiorna);
        myTimer.start();
    }

    private function aggiorna(e:TimerEvent) {
       leggiOra();
       visualizzaOra();
    }
}
```

OROLOGIO

- Dichiaro i metodi inizializzaVisualizzazione e visualizzaOra

```
package {
  public class OrologioGenerico extends Sprite {
  .......
    public function inizializzaVisualizzazione() {
        //metodo da ridefinire nelle classi derivate
    }

    public function visualizzaOra () {
        //metodo da ridefinire nelle classi derivate
    }
}
```

OROLOGIO ANALOGICO

HO IMPORTATO LE CLASSI NECESSARIA

- In rosso la classi che devo ancora aggiungere (mi servono per disegnare il quadrante)

```
import flash.display.Sprite;
import flash.display.Shape;
import flash.display.Graphics;
import flash.utils.Timer;
import flash.events.TimerEvent;

import flash.text.TextField;
import flash.text.TextFieldAutoSize;
```

DICHIARAZIONE DELLA CLASSE

1. Dichiaro la classe OrologioAnalogico facendola discendere da Sprite :

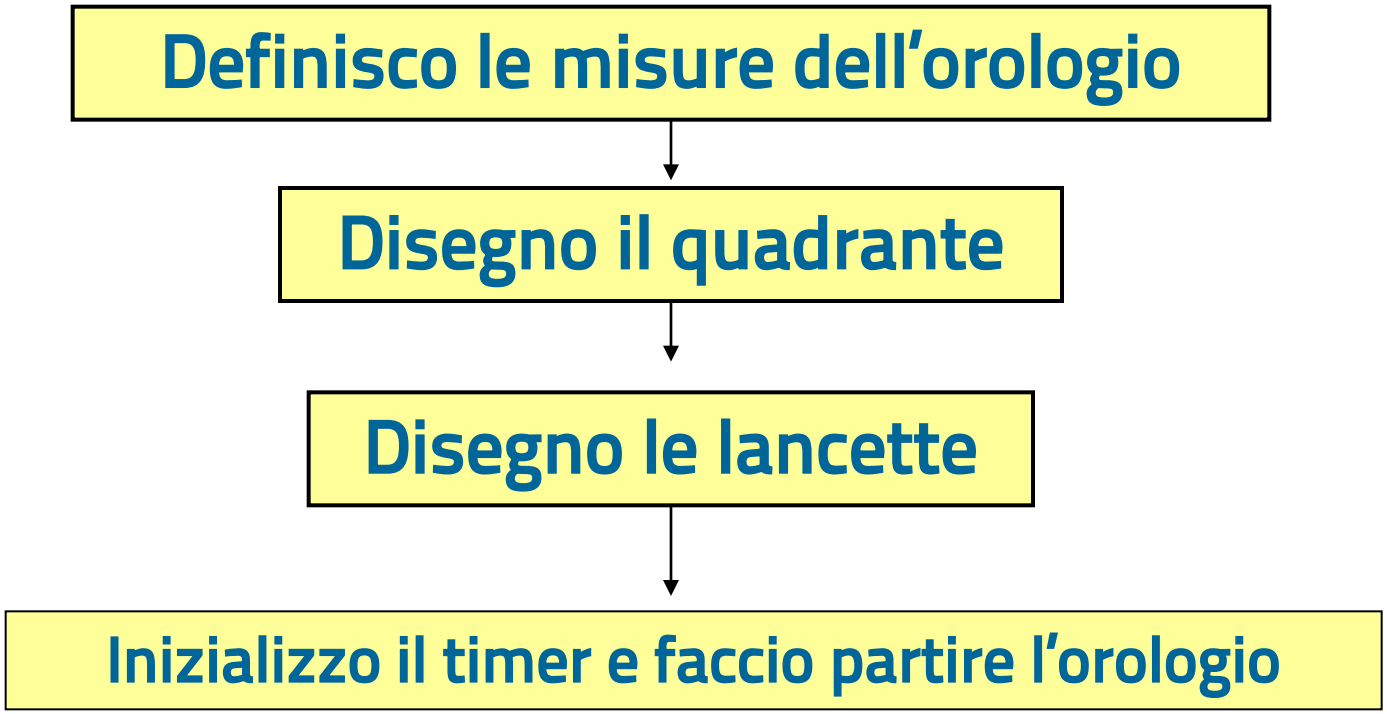
```
package {
  import flash.display.Sprite;
  ......
  public class OrologioAnalogico extends Sprite {
       ......
  }
}
```

HO DICHIARATO DELLE PROPRIETÀ

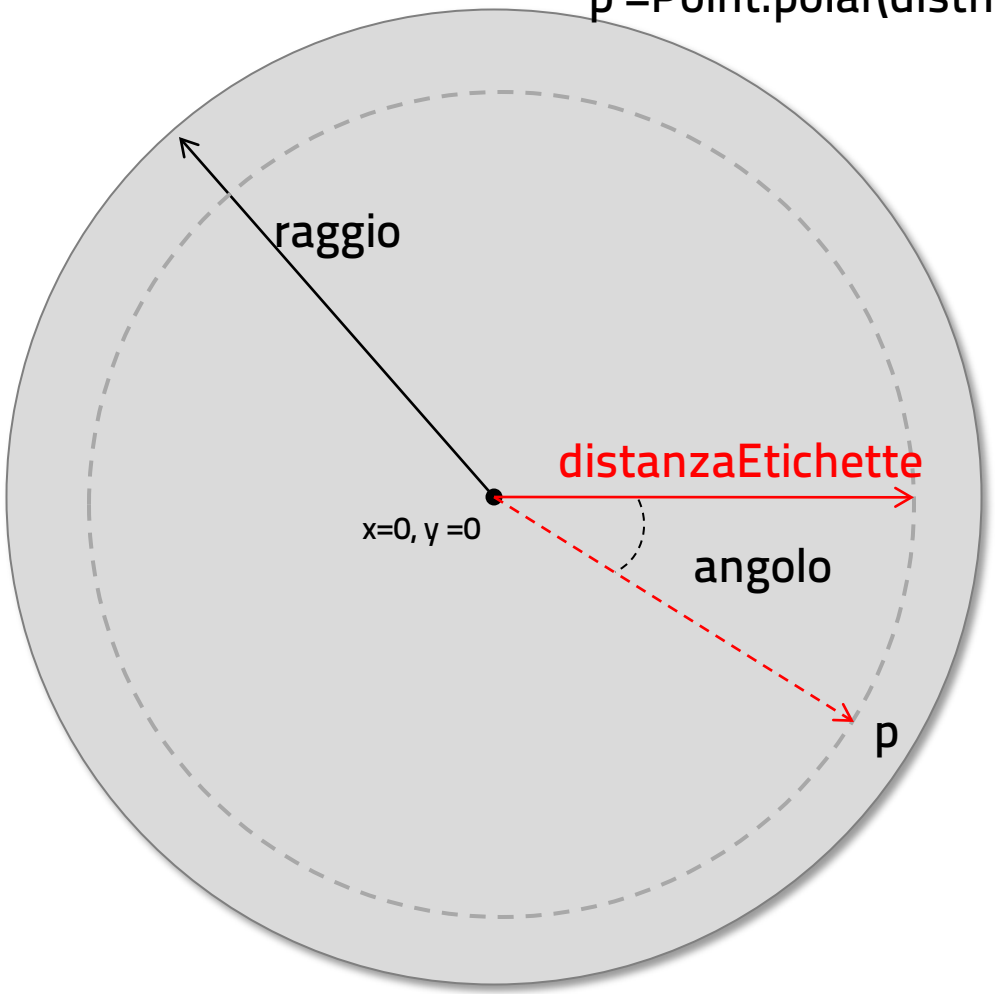
- La definizione di queste misure come proprietà e non come valori rende il mio lavoro più flessibile

```
private var lancettaOre:Shape = new Shape();
private var lancettaMinuti:Shape = new Shape();
private var lancettaSecondi:Shape = new Shape();
private var centrox:Number;
private var centroy:Number;
private var raggio:Number;

private var distanzaEtichette:uint;
```



p =Point.polar(distnzaEtichette, angolo)



quadrante.graphics.drawCircle(0,0,raggio)

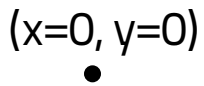
DISEGNARE LA LANCETTA

- Questa funzione disegna una lancetta sulla shape già creata:

```
private function disegnaLancetta(lancetta:Shape,
        colore:uint, lunghezza:uint, spessore:uint){
  lancetta.graphics.moveTo(0,0);
  lancetta.graphics.lineStyle(spessore, colore);
  lancetta.graphics.lineTo(0,lunghezza);
  addChild(lancetta);
  lancetta.x = centrox;
  lancetta.y = centroy;
}
```

DISEGNARE LA LANCETTA

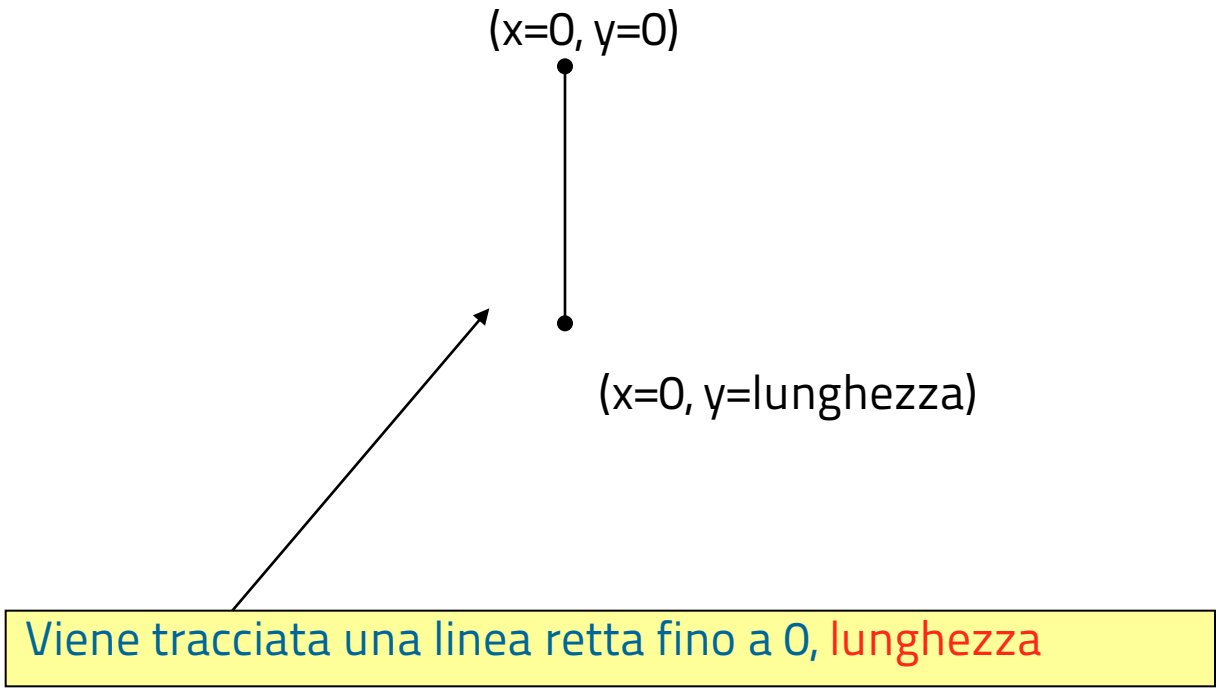
```
lacetta.graphics.moveTo(0,0);
```



Il pennino viene spostate alle coordinate x=0, y=0 relative all'oggetto Shape creato

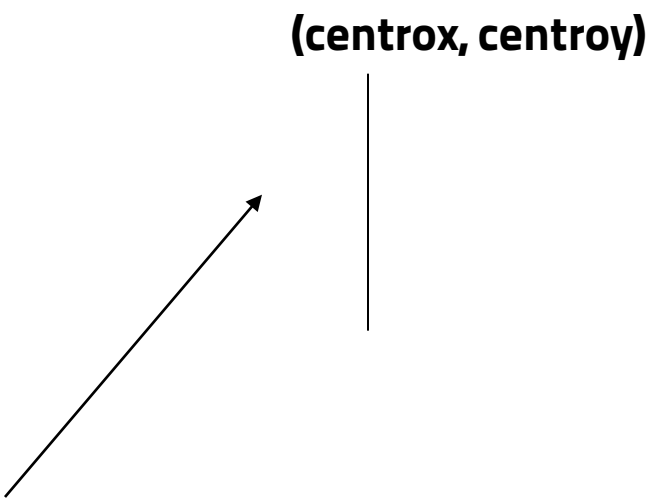
DISEGNARE LA LANCETTA

```
lancetta.graphics.lineStyle(spessore, colore);
lancetta.graphics.lineTo(0,lunghezza);
```



DISEGNARE LA LANCETTA

```
addChild(lancetta);
lancetta.x = centrox;
lancetta.y = centroy;
```



La lancetta viene aggiunta allo schermo e posizionata al centro dell'orologio pronta per essere ruotata dall'evento timer